Universidade de Aveiro
2018

Departamento de Eletrónica, Telecomunicações e Informática

Sara da Silva Furão

CRIAÇÃO DE INTERFACES GRÁFICAS AUTOMATIZADAS, DINÂMICAS E ADAPTÁVEIS

Universidade de Aveiro
2018

Departamento de Eletrónica, Telecomunicações e Informática

Sara da Silva Furão

CRIAÇÃO DE INTERFACES GRÁFICAS AUTOMATIZADAS, DINÂMICAS E ADAPTÁVEIS

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Hélder Troca Zagalo, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri

presidente

Prof. Doutor Joaquim Arnaldo Carvalho Martins
professor catedrático da Universidade de Aveiro

Prof. Doutor Álvaro Pedro de Barros Borges Reis Figueira
professor auxiliar da Faculdade de Ciências da Universidade do Porto

Prof. Doutor Hélder Troca Zagalo
professor auxiliar da Universidade de Aveiro

agradecimentos

Aos meus pais e à minha família, por nunca deixarem de acreditar em mim e me darem força, apoio e incentivo desde o primeiro dia. A vocês, só tenho a agradecer todas as oportunidades e metas que já alcancei assim como a pessoa que sou hoje. O vosso apoio foi fundamental durante todo este percurso.

A todos os meus amigos e colegas, que me acompanharam e apoiaram durante todos os momentos deste percurso académico.

Ao Professor Hélder Zagalo, orientador desta dissertação, por ter acreditado nas minhas capacidades aquando a proposta desta dissertação e pela ajuda, apoio e disponibilidade prestada ao longo deste projeto.

Aos meus orientadores, na Altice Labs, Engenheiros Jorge Monteiro e Jorge Sousa pelos desafios inovadores lançados no início desta dissertação que me motivaram a trabalhar todos os dias para estar à altura dos mesmos e por todas as oportunidades e apoio prestado ao longo da mesma.

Aos meus colegas de equipa, pelo acolhimento, pela disponibilidade, por toda a ajuda prestada, pela partilha de conhecimento, pelos conselhos e por todo o apoio incondicional prestado ao longo deste projeto.

Aos meus colegas estagiários, pela partilha de experiências, conselhos, incentivo e apoio prestado durante esta etapa final deste percurso académico.

A todos, muito obrigada.

palavras-chave

aplicações adaptáveis, interfaces dinâmicas, comportamento do utilizador, inteligência artificial, *machine learning*, *deep learning, deep neural networks, tensorflow*, componentes *web*.

resumo

O ambiente corporativo enquadrado nas telecomunicações conjugado com o aparecimento de novos serviços é cada vez mais complexo e com mudanças/transformações mais aceleradas. O negócio e os sistemas de suporte necessitam de evoluir e adaptarem-se de forma muito rápida para esta nova realidade e exigência. Deste modo, a criação de aplicações adaptáveis e contextualizadas, proporciona interações mais ricas, permitindo que as interfaces entreguem a informação da melhor forma possível, com uma melhor experiência de utilização. Esta dissertação tem como principal objetivo a definição e a implementação de metodologias para gerar interfaces gráficas dinâmicas em tempo real e adaptáveis de acordo com os comportamentos do utilizador. Estas metodologias de Inteligência Artificial, passam pela utilização de técnicas de *Machine Learning*. Os resultados finais permitiram obter uma interface adaptável e dinâmica a cada utilizador, sendo esta gerada em tempo real.

keywords

adaptive applications, dynamic interfaces, user behavior, artificial intelligence, machine learning, deep learning, deep neural networks, tensorflow, web components.

abstract

The corporative environment framed in telecommunications combined with the emergence appearance of new services is increasingly complex and with fast transformations. This new reality and its requirements demand that the business and the support systems evolve and quickly adapt to it. As a result, the creation of an adaptive and contextualized application provides richer interactions, allowing interfaces to offer the information in the best possible way, delivering a better user experience. The main objective of this dissertation is to define and implement methodologies to generate dynamic and adaptable interfaces according to the user behavior, in realtime. These Artificial Intelligence methodologies are developed using techniques of Machine Learning. The final results made it possible to obtain a dynamic and adaptable interface for each user in realtime.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

API	Application Programming Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
DOM	Document Object Model
XML	Extensible Markup Language
GPU	Graphics Processing Unit
HATEOAS	Hypermedia as the Engine of Application State
HTTP	Hypertext Transfer Protocol
IA	Inteligência Artificial
IDE	Integrated Development Environment
IT	Information Technology
JSON	JavaScript Object Notation
MOBI-D	Model-Based Interface Designer
ML	Machine Learning
MVC	Model View Controller
MVVM	Model View View Model
NLP	Natural Language Processing
NPM	Node Package Manager
PWA	Progressive Web Applications
RNN	Recurrent Neural Network
REST	Representational State Transfer
SPA	Single Page Application
UI	User Interface
UM	User Modeling
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
V-DOM	Virtual DOM
WebCT	Web Component Toolkit
WYSIWYG	What You See Is What You Get

1 Introdução

1.1 Enquadramento

O ambiente corporativo enquadrado nas telecomunicações conjugado com o aparecimento de novos serviços é cada vez mais complexo e com mudanças/transformações mais aceleradas. O negócio e os sistemas de suporte necessitam de evoluir e adaptarem-se de forma muito rápida a esta nova realidade e exigência, assim como, o fornecimento ou alteração de aplicações *web* e móveis que têm vindo a conhecer um crescimento exponencial face a estas exigências. Esta massificação de serviços e exigência do negócio tem originado a que os operadores procurem fornecer aplicações distintas e inovadoras, bem como a personalização destas, contextualizadas, ao cliente e ao ambiente onde se inserem. Por isso, são necessárias interfaces de utilizador inteligentes para melhorar a interação entre os utilizadores e estes sistemas.

A criação de aplicações adaptáveis, assim como, o desenvolvimento de mecanismos de recuperação e de filtragem personalizados, proporcionam interações mais ricas e contextualizadas com os utilizadores no seu uso diário, permitindo que as interfaces se ajustem e entreguem a informação da melhor forma possível.

A criação ou alteração manual das interfaces gráficas, representa um grande esforço que se pretende reduzir ou eliminar através de processos automatizados, dinâmicos e com recurso à inteligência artificial, de modo a que estas se adaptem de forma automática e ajustada face às necessidades onde se inserem e para quem se destinam. Atualmente já *existem frameworks low code* que permitem a reutilização de código tornando este processo mais automatizado e dinâmico. O desenvolvimento das interfaces automatizadas visa combinar todos os procedimentos e *frameworks* que permitem a criação e reutilização de componentes *web* genéricos, de modo a que estes possam servir de suporte ao dinamismo necessário ao negócio ou ao utilizador onde se inserem.

Deste modo, pretende-se avaliar as diversas formas possíveis de responder a este desafio, recorrendo aos últimos paradigmas tecnológicos de Inteligência Artificial (IA) tais como a utilização de algoritmos de *Machine Learning* (ML). É ainda requisito que este trabalho seja realizado recorrendo à *framework* de desenvolvimento *web* da Altice Labs, a WebCT.

1.2 Principais Objetivos

Já existem *frameworks* destinadas à criação de interfaces *web* que dão enfase à reutilização de código. Assim, o principal objetivo desta dissertação passa por definir e implementar metodologias para gerar interfaces gráficas dinâmicas em tempo real e adaptáveis de acordo com o utilizador, recorrendo à utilização de algoritmos de *Machine Learning* para deteção de padrões de comportamento dos utilizadores de modo a permitir a criação de uma interface personalizada para cada utilizador.

1.3 Metodologia de Trabalho

Para a realização da dissertação, foi utilizada uma metodologia de trabalho constituída pelos seguintes passos:

1. Estudo das diversas tecnologias:

    - revisão de literatura de *frameworks* já existentes;
    - esforço (homens/dia) para produzir uma interface CRUD;
    - tipo de projetos/aplicações que aplicam;
    - manutenção/evolução;
    - vantagens e desvantagens das *frameworks* de produção de interfaces automatizadas;
    - impactos na rotatividade dos recursos humanos nas empresas;
    - estudo de Angular, Node.js, Java, Javascript, HTML5, *Machine Learning*;
    - estudo da WebCT, a *framework* da Altice Labs para apoio à criação de interfaces *web*.

2. Evolução, de pelo menos, um componente da *framework* WebCT, permitindo que o mesmo se possa ajustar ao comportamento do utilizador.

    - Seleção e familiarização de componentes da WebCT;
    - Identificação de métricas/indicadores associados à sua utilização;
    - Evolução do componente de forma a ajustar-se a diferentes comportamentos de forma dinâmica.

3. Evolução do componente e integração com o sistema de apoio à decisão com base em técnicas de *Machine Learning*.

    - Seleção da tecnologia;
    - Identificar um grupo de utilizadores para o estudo;
    - Identificação de *data frames* e algoritmos a aplicar.

4. Escrita da dissertação contemplando as principais conclusões e análises do trabalho desenvolvido.

1.4 Contribuição

A principal contribuição desta dissertação passou por conseguir criar um sistema que possui dois subsistemas inovadores:

- um dos sistemas explora as capacidades de *Machine Learning* (ML) e como estas podem contribuir para acrescentar valor às aplicações *web*, de modo a criar interações mais ricas com os seus utilizadores. A criação deste sistema exigiu uma exploração sobre o tema de deteção de padrões de comportamento de utilizadores aquando as suas interações com as aplicações e qual a melhor abordagem na utilização dos dados recolhidos, por forma a que modelos de ML possam de facto detetar padrões e ajudar à personalização das interfaces. Foi também necessário explorar *frameworks* em grande ascensão no âmbito das áreas de *Machine Learning* e *Deep Learning*.

- o outro sistema permite adicionar uma característica dinâmica à *framework* WebCT, permitindo que as interfaces sejam geradas no momento antes de serem disponibilizadas. Este processo automatizado e dinâmico permite reduzir ou mesmo eliminar o esforço necessário para criar/atualizar uma interface de forma tradicional (manual).

1.5 Organização da Dissertação

A presente dissertação está dividida em seis capítulos, sendo o presente capítulo destinado ao enquadramento do problema, principais objetivos, metodologias seguidas e à contribuição da solução conseguida nesta dissertação. O segundo e o terceiro capítulo apresentam o estado de arte, descrevendo conceitos teóricos que apoiam a arquitetura proposta. No segundo capítulo são expostos casos de estudo de sistemas adaptáveis e dinâmicos e no terceiro são abordadas plataformas e tecnologias para implementação de interfaces *web*. Neste capítulo é também feita uma comparação entre as várias tecnologias, a fim de perceber a escolha feita. No quarto capítulo são expostos todos os detalhes relativos à solução proposta e desenvolvimento da mesma, apresentando todos os detalhes da arquitetura e implementação. Neste capítulo estão também disponíveis vários diagramas para ajudar a entender todo o processo. No quinto capítulo são apresentados os resultados do sistema da solução implementada, seguindo-se do capítulo seis que expõe as conclusões, problemas encontrados e trabalho futuro. Este documento termina com a bibliografia consultada.

2 Sistemas Adaptáveis e Dinâmicos

Para a compreensão dos conceitos expostos nos próximos capítulos, torna-se necessário conhecer projetos desenvolvidos nesta área e compreender os conceitos subjacentes. Assim, neste capítulo serão apresentados alguns casos de estudo de sistemas adaptáveis já desenvolvidos nesta área e seguidamente serão apresentados alguns conceitos introdutórios aos temas de Inteligência Artificial (IA) e *Machine Learning*, assim como algumas tecnologias especificas utilizadas nestas áreas.

2.1 Conceitos existentes sobre Sistemas Adaptáveis e Dinâmicos

Pat Langley [1], definiu as interfaces adaptáveis como "artefactos de *software* que melhoram a sua capacidade de interagir com os utilizadores construindo modelos de utilizadores com base na experiência parcial com cada um deles".

Posto isto, tendo um conjunto de dados, é necessário criar um modelo computacional para armazenar o perfil do utilizador e desenvolver o mecanismo de adaptação. A "Rede Neural Artificial" representa um desses métodos, sendo uma técnica estatística não-linear, utilizada principalmente para previsão. É um modelo constituído por um grupo de interconexões de neurônios e processos artificiais. Uma rede neural artificial é um sistema adaptativo que altera a sua estrutura, com base em informações externas ou internas que fluem através da rede durante a fase de aprendizagem [2].

Para obter uma interação de sucesso com os utilizadores, as interfaces de utilizador inteligentes necessitam de um mecanismo para reconhecer, caracterizar e prever as ações do utilizador [3], melhorando assim, a interação do utilizador com as interfaces e permitindo interações mais rápidas.

O objetivo de *User Modeling* (UM) consiste em tornar os sistemas de informação mais *user-friendly*, permitindo que estes, adaptem o comportamento do sistema às necessidades do utilizador ao longo do tempo. As técnicas de (ML) foram aplicadas a problemas de modelagem de utilizadores, para adquirir modelos de utilizadores individuais que interagem com um sistema de informação, agrupando-os em estereótipos com interesses comuns [1].

As empresas, cada vez mais se movem para um paradigma centrado no cliente a fim de aumentar a competitividade. O comportamento do cliente não pode ser previsto utilizando métodos analíticos tradicionais, em vez disso, empresas que procuram vantagem competitiva, investigam outro tipo de métodos analíticos baseados em heurísticas e técnicas de IA. Os agentes inteligentes são o próximo nível de abstração em soluções baseadas em modelos para aplicações *business-to-bussiness* [4].

Esta tecnologia de agentes, pode ajudar a enfrentar sérios desafios tecnológicos, tais como preocupações sobre pesquisa eficaz, segurança e privacidade. Para além disso, podem realizar tarefas repetitivas, que nos dias de hoje são realizadas por humanos [4].

Este tipo de agentes, recebem informação diretamente dos utilizadores acerca de como estes organizam e veem as informações, para posteriormente, conseguirem criar um perfil para cada um dos respetivos utilizadores de forma a que a informação já venha organizada e pronta a ser utilizada por estes. Desta forma, oferecem acesso rápido e fácil a serviços especializados relacionados com o perfil (preferência) do utilizador. Em suma, estes agentes "observam" e monitorizam as ações realizadas pelo utilizador na interface e sugerem a melhor maneira de executar as tarefas, ou fazer as melhores recomendações [4].

Técnicas de ML ajudam na previsão de comportamentos e deteção de padrões, que pode levar à automatização de tarefas repetitivas e de longa duração que implicam custos de mão de obra humana. As principais aplicações de aprendizagem com o objetivo de prever diferentes tipos de comportamento humano passam por: recomendações, comportamento, condição individual e previsão de comportamento coletivo [5].

Este tipo de agentes, podem acrescentar grande valor aos vários tipos de portais.

Portais de vendas (exemplo da Amazon), que ajustam as interfaces ao utilizador. Devido à grande oferta, de produtos, que este tipo de portais disponibiliza, os sistemas de recomendação tornam-se indispensáveis para ajudar os clientes/consumidores na difícil tarefa de encontrar os produtos que pretendem comprar ou que estão interessados, evitando assim que estes percam o interesse com pesquisas demoradas. Estes sistemas, conseguem recomendar ou prever uma compra futura com base na aprendizagem "extraída" dos comportamentos anteriores dos clientes. As recomendações tanto sugerem produtos como fornecem informações personalizadas sobre os mesmos, através de técnicas de *cross-selling* e *up-selling* [6].

De certo modo estas técnicas de recomendação, contribuem para a personalização dos portais, porque ajudam os mesmos a adaptarem-se a cada cliente. Oferecendo assim, uma "loja" *online* projetada, adaptada e adequada a cada um. Normalmente estes sistemas agrupam os clientes por segmentos de mercado e através de recomendações pessoais, as empresas são capazes de identificar e antecipar os "desejos" do cliente, porque já "conhecem" outros clientes semelhantes [6].

Os sistemas de recomendação ajudam este tipo de portais a aumentar o número de vendas (principal objetivo dos portais de vendas) e a conquistar a lealdade dos clientes, um fator bastante importante, tendo em conta que outros portais de vendas, encontram-se a um ou dois cliques de distância. Por isso, as

empresas investem nestes sistemas de recomendação para apresentar interfaces personalizadas que atendem às necessidades dos clientes, visto que estes irão aos portais que melhor atendem as suas necessidades e retornarão aos mesmos com os quais mantem uma relação de lealdade *one-to-one*, tratando cada cliente individualmente. Um exemplo de marketing *one-to-one*, é o sistema aprender, que determinado cliente quer sempre que as encomendas sejam enviadas durante a noite, ou por exemplo, que determinado cliente coleciona uma linha inteira de produtos de uma dada marca ou de um dado tipo de produto, por exemplo, relógios [6].

Para além de portais de vendas, os agentes inteligentes podem oferecer vantagens aos portais de *self-management,* que oferecem informação mais orientada ao utilizador fazendo recomendações personalizadas (Google Fit [7], Spotify [8], etc.) e ainda aos portais de operação, que oferecem apoio ao cliente, ajudando e facilitando com as operações a serem feitas. O sistema desenvolvido neste projeto de dissertação foi aplicado a um portal de operação de *backoffice*.

Seguidamente, são mencionados mais exemplos de sistemas de recomendações sobre as previsões de preferências do utilizador. Por exemplo, o Spotify pode gerar uma lista futura de reprodução de músicas favoritas. Este processo passa por prever se a pessoa volta a ouvir a mesma música várias vezes. No caso do Netflix [9], são guardados dados como a classificação de um filme ou série de modo a ser possível, no futuro, prever e recomendar filmes de interesse para o utilizador. A previsão de um conteúdo no qual o utilizador está interessado dá origem à personalização de conteúdos de interfaces, que consiste na otimização automatizada de um serviço. O *website* de vendas, Amazon, com base nos hábitos de compras de cada utilizador, personaliza a sua interface prevendo produtos e *links* que vão de encontro aos desejos do mesmo [5].

As interfaces de utilizador devem tornar-se menos intrusivas e mais inteligentes em relação à adaptação a estes [10]. Posto isto, é percetível que os sistemas adaptáveis se encontram relacionados com técnicas de ML.

A utilização de técnicas de ML em interfaces pode permitir experiências de utilizador altamente personalizadas, mais ricas e contextualizadas. A personalização leva os *designers* a flexibilizar a melhor forma de criar interfaces verdadeiramente adaptativas que são controladas pela lógica de ML [5]. O conhecimento aprendido deve refletir as preferências dos utilizadores, proporcionando individualmente serviços personalizados para cada um [11].

As decisões que o utilizador tem perante a interface, permitem a recolha de dados de treino contribuindo para a aprendizagem por parte do modelo. Ou seja, as interfaces de utilizador adaptáveis, colecionam dados durante a interação com os utilizadores, por isso, o sistema deve realizar a aprendizagem com base no conhecimento que é atualizado sempre que ocorre uma interação com a interface [11]. Normalmente, o modelo vai melhorando consoante o aumento da

duração da interação e por consequência o aumento do tamanho do conjunto de dados recolhidos durante as interações.

Seguidamente, são apresentados mais alguns sistemas sobre vários temas, relacionados com as interfaces de utilizador que apresentam características relevantes de adaptação.

O LiftIgniter é um *software* que tem como objetivo, ajudar a personalizar o conteúdo em qualquer página para um utilizador em especifico, tornando a experiência do mesmo mais agradável e envolvente [12].

A visão deste *software*, passa pelo dever de existir uma API de personalização, onde toda a propriedade digital deve ter a personalização integrada, porque se o objetivo é oferecer aos utilizadores algo que eles desejam, então a única maneira de fazer isso em escala é através da utilização de técnicas de ML [12].

Este *software* analisa uma grande quantidade de dados que definem um perfil de utilizador, tomando decisões em tempo real para cada um. Todas as ações do utilizador devem implicar uma reação inteligente, dinâmica e perfeita por parte do *website* [12]. Este, é automatizado, estando sempre em "personalização" porque está constantemente a aprender e a otimizar dinamicamente [13].

Em suma, não é possível atualizar cada *website* manualmente de modo a atender às necessidades dos seus visitantes específicos, especialmente para cada utilizador individual, por isso, o LiftIgniter oferece um mecanismo que suporta ML de personalização e recomendação para qualquer *website* ou aplicação [13].

O sistema *Adaptive diagnostics and personalized technical support* fornece um suporte de desempenho eletrônico, inteligente e adaptável para a manutenção de equipamentos complexos. Tem uma caracterização dinâmica com experiência e preferências de um técnico sob a forma de um modelo de utilizador, ajustando a sua estratégia de diagnóstico de acordo com o técnico que está a utilizar o sistema naquele momento e que tarefas é que ele está a realizar, adaptando dinamicamente, as sequências de configurações, testes, reparações ou procedimentos de substituição com base nas respostas do técnico [14],[15].

No artigo [16], para técnicas de ML são utilizadas as seleções de *widgets* para gerir as diretrizes de adaptação.

No artigo [17], é descrito um sistema que aplica um algoritmo adaptativo ao *design* automatizado da interface de utilizador dentro da estrutura do ambiente de desenvolvimento da interface MOBI-D (*Model-Based Interface Designer*). No mesmo artigo é mencionado a *Inference Bear* que é uma aplicação de programação por demonstração que infere especificações de *design* gerados

pelo utilizador. O *Inference Bear* usa a adaptação para gerar uma interface de utilizador personalizada observando o comportamento do mesmo [17].

O artigo [3], sublinha que técnicas de ML foram aplicadas com sucesso à previsão das ações de um utilizador. Deste modo, ML promete enfatizar novos cenários para adaptação ao nível de interface de utilizador [16].

As interfaces inteligentes, são interfaces que "tentam aprender" quais são os interesses de um utilizador em específico, e para isso, é importante "observar" o comportamento do mesmo [27], de modo a ser possível recolher dados que permitam compreender a existência de um padrão de comportamento. Assim, é importante entender e estudar quais são os *inputs* mais relevantes que podem permitir interpretar o "perfil" e identificar o padrão de comportamento do utilizador.

Existem muitos estudos sobre agentes inteligentes que podem "aprender" quais são os interesses de um utilizador. Por exemplo, dado um utilizador específico e uma determinada página, o agente deve ser capaz de prever o nível de interesse do utilizador naquela página. Se o agente detetar que o utilizador tem interesse nessa página, este pode identificá-la e sugeri-la posteriormente ao utilizador num outro momento [27]. Existem vários métodos e algoritmos que podem ser utilizados para a recolha de exemplos de dados de treino. Nos artigos [28][29], foi realizada uma análise ao comportamento de navegação de um utilizador, observando as ações que este realiza na página (*links* selecionados, *scroll* e a atividade do rato).

Os *inputs* para deteção de ações numa página passam por analisar os *inputs* do rato e do teclado. Por exemplo, a posição do cursor do rato, as teclas pressionadas e a dimensão da janela do navegador [30]. Através do conhecimento sobre o *layout* das páginas HTML, é possível saber todas a atividades realizadas pelo utilizador, na página [30] e através do posicionamento do cursor e do *scroll*, é possível perceber quais as partes da página que o utilizador tem mais interesse. Através do número de cliques com o rato, é possível detetar quais são os elementos da página que são mais utilizados pelo utilizador, identificando assim os elementos que tem maior interesse para este.

Outro exemplo, no caso de um *widget* tabela, é possível identificar a coluna com mais interesse para o utilizador (através do número de cliques) e alterar a ordem das colunas, colocando em maior destaque as que representam maior interesse para o utilizador. Se o utilizador tiver problemas ou dificuldades visuais, vai ter tendência a aumentar a dimensão da janela de navegação, sendo possível interpretar a acessibilidade à página para um determinado utilizador.

Após a identificação dos *inputs* relevantes ao problema que se pretende resolver, e após a recolha dos dados recolhidos das interações que os utilizadores tem com os mesmos, é importante perceber quais as melhores técnicas capazes de adicionar "inteligência" às interfaces *web*.

2.2 Inteligência Artifcial, *Machine Learning* e *Deep Learning*

Tendo um conjunto de dados, e recorrendo aos últimos paradigmas tecnológicos na área de Inteligência Artificial, tais como a utilização de algoritmos de *Machine Learning* e *Deep Learning*, é possível prever as melhores opções e decisões relativas a componentes e às suas características de modo a oferecer a interface mais adaptável para cada utilizador. Primeiramente é importante entender bem os conceitos de Inteligência Artificial, *Machine Learning* e *Deep Learning*.

A Inteligência Artificial é a ciência que torna as máquinas "inteligentes". *Machine Learning*, é o ato de programar as máquinas para que estas consigam, através de algoritmos aprender e que consigam atribuir significados às informações que recebem e que sejam capazes de posteriormente realizar previsões em novos dados. *Deep Learning* (DL) consiste na criação de uma rede neural artificial, que permite à máquina uma semelhança cognitiva com o cérebro humano a fim de interpretar e interligar os dados [18]. Em suma, estes três conceitos encontram-se inevitavelmente relacionados (Figura 1), pois, ML é um ramo da IA e DL é uma forma de ML que permite que as máquinas aprendam com a experiência e compreendam o mundo em termos de uma hierarquia de conceitos [19].

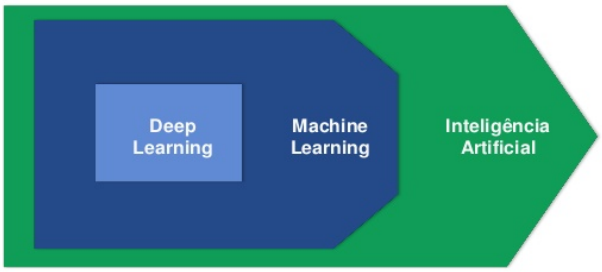

Figura 1 - Inteligência Artificial, Machine Learning e Deep Learning [18]

Inteligência Artificial

Segundo *www.dictionary.com*, "inteligência" é a capacidade de adquirir e aplicar conhecimento, a capacidade de pensar e raciocinar e o conjunto de capacidades superiores da mente. Em 1956, John McCarthy designou a Inteligência Artificial como a ciência e a engenharia que torna as máquinas inteligentes, especialmente programas de computador inteligentes [20].

A tabela seguinte, indica as direções seguidas por Russell e Norvig, em 1995:

Sistemas que **pensam** como os humanos	Sistemas que **pensam** racionalmente
Sistemas que **agem** como humanos	Sistemas que **agem** racionalmente

Tabela 1 - Inteligência Artificial [19]

Em 1950, no seu artigo *"Computing Machinery and Intelligence"*, Alan Turing, um matemático britânico, introduziu o "Teste de Turing" (Figura 2) como resposta à questão "As máquinas podem pensar?". Quando numa conversa, um humano, via terminal, com dois interlocutores fisicamente ausentes, não for capaz de distinguir a "criatura" biológica da artificial, então é porque a criatura artificial é inteligente e diz-se que a máquina passou no teste [21]. O "Teste de Turing" representa o grande desafio da IA, avaliando o quão próximas as respostas dadas pela máquina são das respostas dadas por um humano.

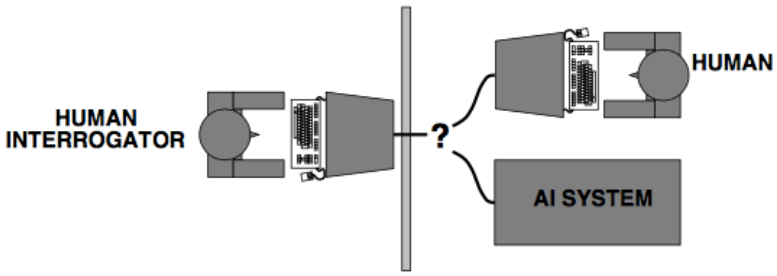

Figura 2 - Teste de Turing [21]

Existe um conjunto de tecnologias que quando combinadas fazem a IA funcionar tais como: *Machine Learning* (ML), *Deep Learning* (DL) e Processamento de Linguagem Natural (NLP). NLP é uma tecnologia que utiliza a ciência da computação para estudar linguagens humanas, que inclui análise sintática e semântica, extração de informação, *text mining*, tradução automática, recuperação de informação, sistemas de perguntas e respostas e sistemas de diálogo [22].

Atualmente a IA, já é utilizada por um grande número de pessoas. Por exemplo:

- aplicação *Siri* disponível em todos os dispositivos Apple (iPhone, iPad, Mac, etc.), utiliza um processamento de voz para atuar como assistente pessoal para responder a questões (por exemplo, "Quem ganhou a final da NBA em 2018?"), fazer recomendações ("Encontra-me uma boa pizzaria nas redondezas") e executar ações (por exemplo, "Liga para mãe", "Tira uma *selfie*");

- a rede social Facebook, utiliza o reconhecimento de imagem para recomendar a identificação de pessoas em fotografias;

- o *website* de vendas *online*, Amazon, faz recomendação personalizada de produtos aplicando algoritmos de ML;

- o *browser* Google consegue preencher automaticamente as pesquisas utilizando recursos de inteligência artificial e também prevê o que se pretende pesquisar com grande precisão.

A IA é utilizada para criar sistemas inteligentes e *Machine Learning* permite prever o comportamento de um utilizador com base no conhecimento adquirido sobre os seus comportamentos anteriores [23].

Machine Learning

O conceito de *Machine Learning* (ML) consiste num subcampo da Inteligência Artificial (IA) que surgiu em consequência da evolução do estudo de reconhecimento de padrões e da teoria da aprendizagem computacional em IA.

Hoje em dia, é utilizado em diversas aplicações, sendo um conceito central para sistemas inteligentes, levando à introdução de tecnologia inovadora e conceitos mais avançados de pensamento artificial [24]. Por exemplo, quando se acede a um determinado anúncio, as próximas publicidades têm como tema, o tema do anúncio pesquisado anteriormente [25], isto é possível porque a máquina é capaz de identificar o perfil de um utilizador através de um estudo realizado ao comportamento do mesmo.

Para ajudar a máquina a compreender o ambiente virtual foram criados algoritmos que operam construindo um modelo a partir de *inputs* de amostras [24]. Seguidamente, a máquina toma uma determinada decisão e executa tarefas especializadas. O verdadeiro desafio é projetar o algoritmo e utilizá-lo da melhor maneira, porque é muito importante que os algoritmos sejam otimizados e de baixa complexidade porque quanto mais eficiente for o algoritmo, mais decisões eficientes tomará.

Tom M. Mitchell [26], forneceu uma definição mais formal de ML, "Diz-se que um programa de computador aprende pela experiência **E**, com respeito a um tipo de tarefa **T** e performance **P**, se a sua performance **P** nas tarefas **T**, na forma medida por **P**, melhoram com a experiência **E**".

A área de ML fornece metodologias adequadas para a tarefa de "extrair conhecimento" de dados [27], ou seja, estas metodologias aprendem padrões dos dados suscetíveis de previsão para posteriormente tomar decisões otimizadas. Também os seres humanos antes de "adivinharem" ações futuras, baseiam-se em fatos do momento e em experiências passadas, e utilizando

estas informações tentam prever o futuro [3]. Estes são os conceitos principais que os agentes preditivos utilizam, assim como os humanos.

A escolha dos algoritmos para o modelo depende de vários fatores, tais como o tipo de dados, dimensão dos dados, etc. Para saber se um modelo aprendeu, deve-se definir uma medida de sucesso. Esta medida geralmente não é baseada no quão bem o modelo executa as experiências de treino, mas sim no quão bem o modelo executa em novas experiências.

Uma parte fundamental da avaliação de modelos, consiste na separação de dados em conjuntos de treino e de teste, visto que para um modelo obter bons resultados, as previsões devem ser aplicadas a dados que são anteriormente desconhecidos [27]. A técnica *holdout* consiste em dividir o conjunto de dados, utilizando 70% dos dados para a fase de treino e os restantes 30% para a fase de teste [28], onde o conjunto de treino é um conjunto de dados utilizado para construir o classificador (modelo), e quanto maior for este conjunto, melhor é o classificador obtido, aumentando a probabilidade de assertividade na fase de teste. O conjunto de teste é utilizado para estimar o desempenho do classificador, avaliando a precisão e a eficácia do modelo [29].

Em seguida são indicados os modelos matemáticos mais comuns de ML.

- Aprendizagem Supervisionada

Primeiramente, com base num conjunto de dados é realizado um treino sobre os mesmos e posteriormente quando o programa receber novos dados já será capaz de tomar decisões precisas [30]. Este modelo contém várias subcategorias:

- Classificação

A classificação consiste na atribuição de um "rótulo"/classe a uma entrada. É geralmente utilizada quando a previsão tem uma natureza distinta, por exemplo, classificar uma dada entrada em "sim" e "não" [30]. O objetivo principal da classificação é identificar o conjunto de categorias a que uma nova observação pertence. Este processo, é realizado com base num conjunto de treino composto por instâncias que já são rotuladas como as classes conhecidas [31].

Seguidamente, são indicados alguns exemplos da utilização de classificação para determinadas tarefas como [32]:

- deteção de rostos, identificação de pessoas em imagens, reconhecimento de expressões faciais;
- identificação de objetos em imagens;
- reconhecimento de gestos em vídeo;

- deteção de vozes, tradução do discurso, etc;
- classificação de *emails* de *spam* ou fraudulentos.

- Regressão

O algoritmo de regressão é similar ao de classificação, mas restringido a valores numéricos.

➢ Aprendizagem Não Supervisionada

O algoritmo de aprendizagem não tem uma fase de treino, só recebe o conjunto de dados de entrada e com base neste, deve descobrir um padrão ou as relações entre os dados, agrupando os objetos, levando em consideração uma certa métrica de similaridade [31].

- *Clustering*

A técnica de *clustering* permite dividir os dados em novas categorias ou subgrupos, exibindo as relações entre estes.

A figura seguinte, exibe de forma esquematizada os modelos e algoritmos de ML.

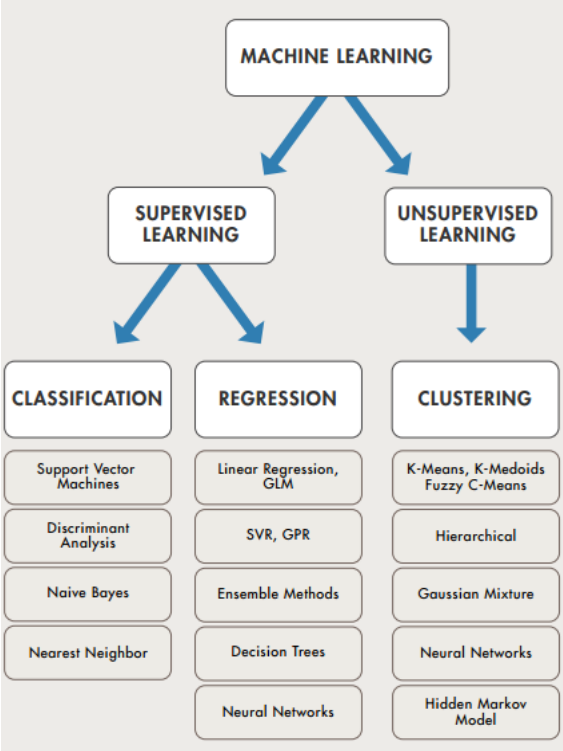

Figura 3 - Algoritmos de Machine Learning [31]

Existem bibliotecas que facilitam computacionalmente a aplicação destes algoritmos de ML.

Deep Learning

O conceito de *Deep Learning* surge da criação de uma rede neural artificial que permite à máquina uma semelhança cognitiva com o cérebro humano a fim de interpretar e interligar dados, de forma a permitir que a rede adquira conhecimento através de um processo de aprendizagem [33].

Um modelo de DL consiste numa rede neural multicamada, composta por várias camadas que processam os dados em estilo hierárquico. Na etapa *feedforward*, cada camada recebe um *input* e produz um *output* que será o *input* da próxima camada, e assim consecutivamente até à última camada [34]. A informação segue sempre uma direção.

A utilização de redes neurais veio facilitar tarefas como reconhecimento de imagens, reconhecimento de voz, encontrar relações e padrões em conjuntos de dados, etc. Contrariamente a algoritmos de DL, a maioria dos algoritmos de ML em conjuntos de dados com muitas variáveis, tendem a obter piores resultados no valor da *accuracy* do modelo [33].

Uma **rede neural** é definida como um grafo de camadas, onde uma camada é uma composição de operadores matemáticos. Uma camada completamente conectada, multiplica os seus dados de *input* por uma matriz de peso, que são utilizadas para armazenar o conhecimento adquirido [33] e adiciona um vetor (*bias*), aplicando por fim uma função de ativação ao resultado. Este procedimento pode ser observado, na Figura 4.

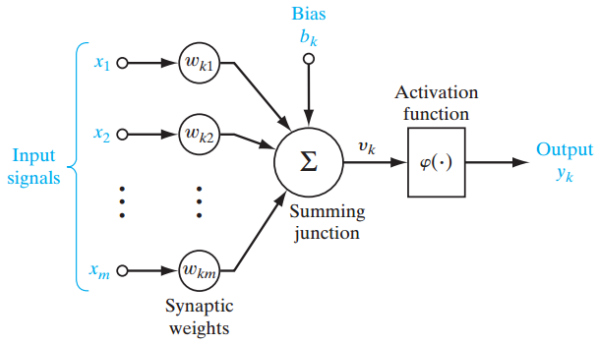

Figura 4 - Camada de uma rede neural [32]

A inicialização dos parâmetros, pesos e *bias* é muito importante na obtenção do modelo final e a estratégia de inicialização destes deve ser selecionada de acordo com a função de ativação escolhida [33]. A função de ativação define o valor de saída de um neurónio.

A figura seguinte é a representação de uma rede neural simples, com três neurónios (nós) de entrada, uma camada escondida (*hidden layer*) e dois neurónios de saída.

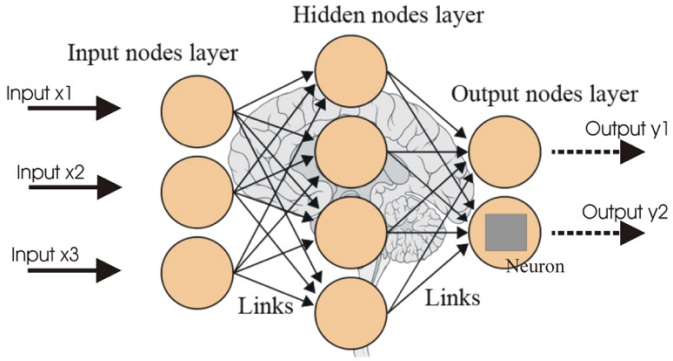

Figura 5 - Rede Neural [32]

Uma função de perda (*loss function*) é uma função escalar que quantifica a diferença entre o valor previsto (para um determinado ponto de dados de *input*) e o valor real, retornando o valor de erro. A etapa *backpropagation* tem como objetivo minimizar esse erro. Numa camada totalmente conectada, a matriz de peso e o vetor *bias* são parâmetros, que o algoritmo poderá atualizar para minimizar o valor da função de perda, de modo a reduzir o erro obtido. A etapa de treino da rede consiste em aplicar a etapa de *backpropagation* e *feedforward* para cada dado [35].

2.3 Tecnologias de *Machine Learning* e *Deep Learning*

Nesta secção são apresentadas algumas *frameworks* de ML e DL.

Scikit-learn

A Scikit-learn, é uma biblioteca de ML, *open-source*, em linguagem Python. É útil, para problemas de aprendizagem supervisionada e aprendizagem não supervisionada para tarefas de pequena e media escala [36], [37]. Permite implementar vários algoritmos de ML, pré-processamento de dados, validação de modelos, e fornece ferramentas de visualização. No entanto não é tão direcionada para redes neurais e não é eficiente para incorporar com GPU.

Torch

A Torch, é uma *framework* para criação de modelos de DL. O seu núcleo é implementado em C/CUDA e a interface em linguagem script Lua14 [36]. Fornece um ambiente semelhante ao Matlab para DL, que providência uma ampla variedade de algoritmos de ML. É utilizada para aplicações de ML em larga escala, como por exemplo, aplicações de fala, vídeo de imagem, etc.

Oferece suporte a CPU e GPU. Utiliza unidades como, *arrays* multidimensionais e tensores. Disponibiliza modelos pré-treinados [38].

Theano

A Theano, é uma biblioteca para criação de modelos de DL, em Python, convertendo posteriormente para C++/CUDA. Assim como a TensorFlow, é baseado em grafos computacionais, no entanto a sua *framework* de visualização não é superior ao TensorBoard [36], a ferramenta de visualização da Tensorflow. As redes neurais são representadas como matrizes e vetores, e todas as operações são definidas como cálculo de matrizes. A Theano permite personalizar todos os parâmetros de uma rede, desde os nós, as camadas, funções de ativação e a percentagem de dados de treino [38].

Caffe

A Caffe, é uma *framework* de DL, implementada em C++ e Python e utiliza redes neurais como bloco básico de construção computacional. Foi desenvolvida para classificação de imagens recorrendo a *Convolutional Neural Networks* (CNN) [36]. Uma das vantagens da Caffe é que não é necessário escrever código, para a construção de modelos, estes são definidos em estruturas de texto simples. Através do "Modelo Zoo", fornece um conjunto de modelos pré-treinados, que facilita a implementação dos algoritmos [38].

TensorFlow

A TensorFlow, é uma *framework* inovadora de *software*, *open-source*, que realiza computação numérica utilizando grafos de fluxos de dados. Permite construir e desenvolver modelos de ML e DL, particularmente utilizando redes neurais [39].

Os nós no grafo representam operações matemáticas e a comunicação entre os nós é representada através de tensores, que são *arrays* de dados multidimensionais [40]. Os nós executam operações matemáticas [35].

A figura seguinte, representa um grafo com tensores e nós que aplica a seguinte operação matemática: "result = x+y".

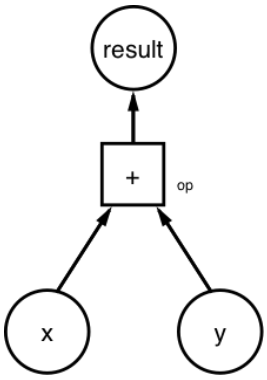

Figura 6 - Grafo que executa x+y [39]

Esta *framework*, apresenta uma arquitetura flexível, projetada para ser escalável em vários computadores, bem como para um ou mais CPUs e GPUs dentro de máquinas individuais. Se se pretender utilizar computação paralela para obter melhor desempenho por parte do algoritmo, a TensorFlow recomenda a instalação do *software* CUDA Toolkit produzido pela NVIDIA, que utiliza os "poderes" da unidade de processamento gráfico (GPU) para realizar algumas operações mais rapidamente [41], [42].

Apesar de o núcleo da TensorFlow ser implementado em C++, a API principal é em linguagem Python. Suporta também APIs para linguagens como C++, Java, Go e JavaScript [43].

Originalmente desenvolvida com o propósito de conduzir ML para pesquisas de *Deep Neural Networks* [40], a TensorFlow é utilizada para definir modelos de ML e de DL e treinar os mesmos com dados. Também é utilizada para processamento de linguagem natural, inteligência artificial, visão computacional e análise preditiva [30].

A Google utiliza esta *framework* em vários serviços que disponibiliza como o Gmail, sistemas de reconhecimento por voz, Google Fotos, Pesquisa Google, etc [35].

A TensorFlow é uma estrutura de baixo nível, por isso, os utilizadores necessitam saber programar e entender os conceitos necessários sobre treino de modelos de redes neurais, no entanto existe Keras e Pretty Tensor, que já tem uma estrutura de alto nível, que simplificam a construção de redes neurais [34]. A TensorFlow também disponibiliza uma API de alto nível de estimadores.

Os grafos dos modelos podem tornar-se bastante complexos, por isso a TensorFlow disponibiliza a TensorBoard, uma *framework* de apoio, que permite visualizar os grafos criados, de modo a facilitar a compreensão da estrutura dos mesmos. Para além disso, pode exibir informações sobre sumários e alterações ao longo do processo de treino do modelo e ver como este evolui [35]. Por exemplo, permite acompanhar a alteração dos parâmetros e da performance do modelo durante as interações de treino [34].

A TensorFlow utiliza dois tipos de tensores: *placeholders* e variáveis. Os grafos só recebem tensores do tipo *placeholders*, que não precisam ser inicializados quando declarados, porque vão ser "alimentados" pelo conjunto de dados de treino em *runtime* enquanto que os tensores do tipo variáveis precisam ser inicializados quando declarados. Tensores do tipo variável, são tensores que são armazenados, em vez de serem eliminados depois de utilizados, pelo grafo. Os pesos e o *bias* são tensores do tipo variáveis que são armazenadas e que vão sofrer atualizações durante as interações do processo de treino do modelo [39].

Exemplo de Regressão Linear utilizando TensorFlow

Apesar de a TensorFlow ser mais direcionada para modelos que utilizam redes neurais, também permite a criação de modelos mais simples. Seguidamente, é exemplificado a criação de um modelo de regressão linear utilizando TensorFlow. Este tipo de problemas procura uma relação linear entre duas variáveis.

Exemplo: Pretende-se saber qual será a previsão da quantidade de produtos vendidos quando o valor unitário for 2,0€ (Tabela 2).

Ano	Custo	Quantidade
2018	1,62	159
2017	1,66	160
…	…	…

Tabela 2 - Dados de Problema de Regressão Linear

Sendo a equação da regressão linear `Y = W X + b` :

```
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.int32)

W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

pred = tf.add(tf.multiply(X, W), b)
```

Primeiramente, são declarados quatro tensores, dos quais X e Y são do tipo *placeholder* que vão receber respetivamente dados do tipo decimal e inteiros. O tensor **X**, vai receber os valores dos custos e o tensor **Y**, os valores das quantidades. O tensor ***W*** e o tensor **b**, são tensores do tipo variável que representam respetivamente o peso e o *bias*. Neste exemplo, W e b são inicializados com um valor *random*, no entanto existem funções de ativação como sigmoid e reLu que são mais adequadas para inicializar estes híper parâmetros. Apesar de a função *sigmoid* ser amplamente utilizada como função de ativação em modelos de redes neurais, as funções *hyper* tangente e ReLu são mais adequadas para modelos de DL [44]. A função sigmoid é uma função

logística limitada entre 0 e 1, a função *hyper* tangente permite que as funções ativação variem de -1 a +1 e a função ReLu de 0 a + ∞ [33].

Seguidamente, são utilizadas algumas operações matemáticas disponibilizadas pela TensorFlow. Onde **tf** significa a instância sobre a TensorFlow e "*add*" e "*multiply*" o tipo de operações matemáticas. Multiplica-se primeiro o valor de X (custo = 2) pelo W (peso) e depois adiciona-se o *bias*. Obtém-se assim a quantidade a prever quando o custo é de 2,0€. Se o valor previsto é diferente do valor que deveria ter sido obtido, então é necessário ajustar o valor do peso e do *bias* de modo a reduzir o erro e a obter a reta otimizada. As figuras seguintes representam respetivamente, um possível gráfico deste problema, em que a reta simboliza a reta otimizada e os pontos, o valor de erro e um grafo representativo da equação de regressão linear, `Y = W X + b`.

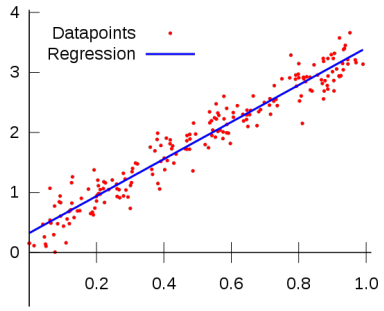

Figura 7 – Gráfico de Regressão Linear [44]

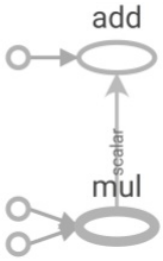

Figura 8 - Grafo de Equação de Regressão Linear

Em modelos mais complexos a TensorFlow utiliza redes neurais. Onde os nós, correspondem aos neurónios da rede e os tensores que contem os dados vão fluindo pelos neurónios. Cada neurónio executa operações matemáticas.

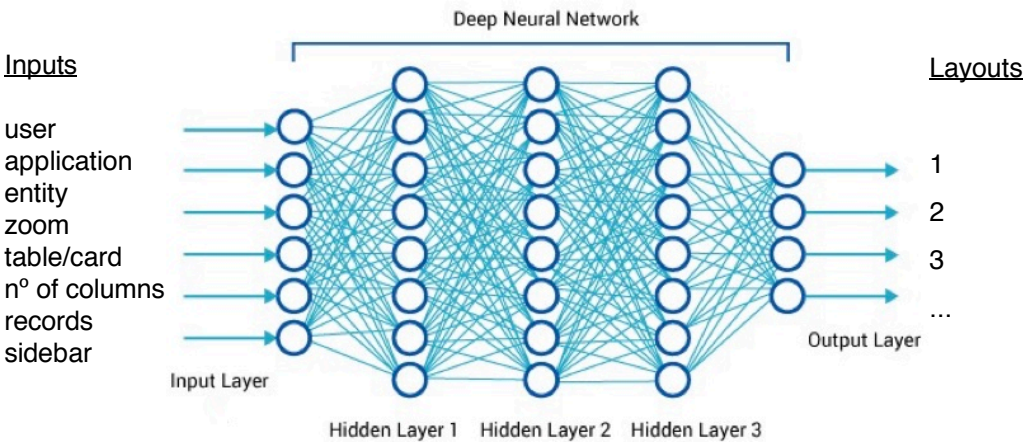

Figura 9 - *Deep Neural Network* [43]

Para o caso de uso em estudo (Figura 9), o objetivo é prever o *layout* mais adequado para cada utilizador. A camada de entrada vai receber os dados recolhidos durante cada interação entre os utilizadores e a aplicação. Esses dados correspondem à aplicação que está a ser utilizada, ao utilizador, à entidade, ao valor de zoom, *widget* selecionado (tabela ou *card*), número de colunas da tabela, número de registos por página da tabela e o estado do *sidebar* de filtragem se está aberto ou fechado. Para cada interação recebida na camada de entrada é produzido um único valor apenas pela camada de saída. Esse valor corresponde ao id do *layout* atribuído pelo modelo para aquela interação.

Processo de Estimativa no TensorFlow

Segundo o dicionário de Oxford, um estimador, é uma regra, um método ou um critério para chegar a uma estimativa do valor de um parâmetro. Esta definição vai de encontro à definição de previsão, que segundo o dicionário, é estimar algo que irá acontecer no futuro.

Apesar de a TensorFlow ser uma API de baixo nível, fornece uma API de alto nível de estimadores que disponibiliza modelos pré-contruídos para treinar e prever (estimar) dados. Esta API, permite simplificar a programação de construção de modelos de ML [45].

Um estimador recebe os dados retornados de uma função de entrada (*Input Function*) definida pelo programador e é configurado pelo modelo fn, uma função que constrói um grafo TensorFlow e é responsável por colocar os dados de *input* no estimador. Retorna as informações necessárias para treinar um modelo, testá-lo e fazer previsões [37].

Como a figura seguinte indica, um estimador tem três métodos principais: treinar, testar e prever (Figura 10). Cada método, recebe uma função de entrada (*Input function*) como argumento.

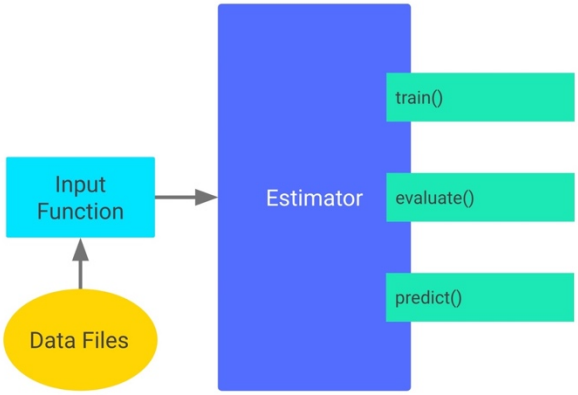

Figura 10 - Fluxo de um Estimador [46]

A TensorFlow permite a utilização de estimadores pré-construídos e também permite a criação de estimadores personalizados, em que é possível configurar todos os parâmetros do modelo. Como a seguinte imagem indica, ambos, são baseados na classe tf.estimator.Estimator.

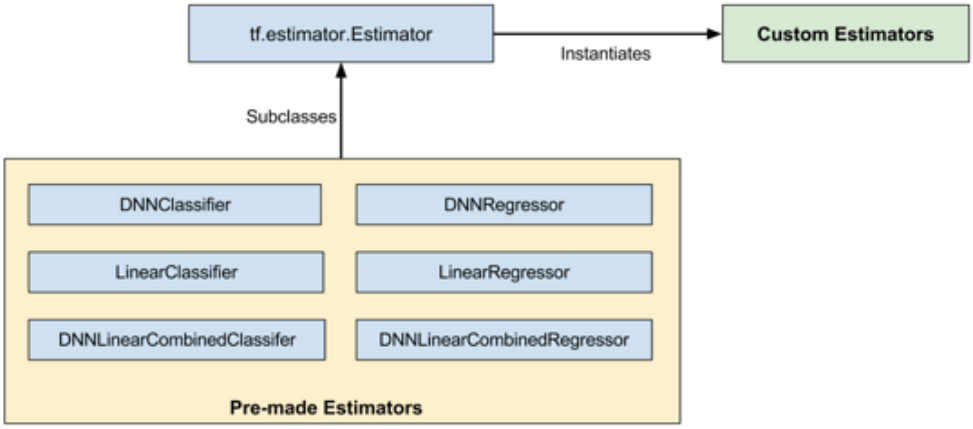

Figura 11 - Conjunto de Estimadores pré-construídos [47]

Na imagem acima (Figura 11), é possível verificar que os estimadores pré-construídos são subclasses da classe base tf.estimator.Estimator e os estimadores personalizados, são uma instância dessa classe. A TensorFlow providencia vários estimadores pré-construídos, dos quais [48]:

- **tf.estimator.LinearClassifier**: para classificadores baseados em modelos lineares;

- **tf.estimator.LinearRegressor**: para problemas de regressão baseados em modelos lineares;

- **tf.estimator.DNNClassifier**: treina modelos de classificação por meio de redes neurais densas e avançadas;

- **tf.estimator.DNNRegressor**: treina modelos de regressão por meio de redes neurais densas e avançadas;

- **tf.estimator.DNNLinearCombinedClassifier**: para modelos "*wide & deep"*, sendo útil para problemas de classificação em grande escala, utilizando um modelo linear e uma rede neural para treinar o modelo;

- **tf.estimator.DNNLinearCombinedRegressor**: para modelos "*wide & deep"*, sendo útil para problemas de regressão em grande escala, utilizando um modelo linear e uma rede neural para treinar o modelo.

Comparação de Tecnologias de *ML* e *DL*

Em seguida, é apresentada uma tabela síntese sobre as principais tecnologias de DL e de ML.

Característica	Sckit-learn	Torch	Theano	Caffe	TensorFlow
Linguagem	Python	Lua, Python, MatLab	Python	C++, Python, MatLab	Python, C++, JAVA, Go, etc.
Proprietário	INRIA, Telecom, ParisTech e Google	NYU by Ronan Collabert	University of Montreal	Facebook	Google
Computação Paralela	-	Multi GPU	Multi GPU	Multi GPU	Multi GPU, Multi node
Arquiteturas DNN	-	CNN, RNN, etc.	CNN, RNN, etc.	CNN	CNN, RNN, etc.
Computação em Grafo	Não	Sim	Sim	Não	Sim

Tabela 3 - Tabela síntese de tecnologias de ML e de DL

A Tabela 3, é referente à síntese de algumas tecnologias de ML e de DL. Relativamente à linguagem, Python é a mais utilizada pela maioria das tecnologias. Sendo que Torch, Caffe e Tensorflow oferecem mais que um tipo de linguagem. Torch, Theano, Caffe e Tensorflow permitem incorporar computação paralela (GPU). Relativamente às arquiteturas de *Deep Neural Networks*, a mais suportada é a *Convolutional Neural Network* (CNN) e as *Recurrent Neural Network* (RNN). As CNN, são muito utilizadas no campo de visão computacional, como classificação de imagens e reconhecimento de objetos em fotografias e vídeos. As RNN, são muito utilizadas em tarefas relacionadas ao processamento de linguagem natural, como modelagem e marcação de sequências, reconhecimento de entidades, análise de sentimentos, tradução de textos, etc.

Por fim, Torch, Theano e TensorFlow são baseadas em computação em grafo.

3 Plataformas e Tecnologias para Implementação de Interfaces *Web*

O presente capítulo, introduz o conceito de *frameworks low code* e de *frameworks* orientadas a componentes *web*, apresentando algumas das tecnologias mais conhecidas e utilizadas no mercado. Primeiramente são apresentadas várias *frameworks low code* disponíveis no mercado e também é apresentada a *framework low code* de desenvolvimento *web* da Altice Labs, a WebCT. Seguidamente, é introduzido o conceito de componente e são mencionadas algumas *frameworks* e bibliotecas orientadas a componentes *web*. O facto de serem orientadas a componentes, permite uma manipulação maior e mais ágil sobre a composição das páginas *web*. Devido à flexibilidade deste tipo de tecnologias, estas tornam-se uma possível solução viável para originar a criação de uma interface dinâmica em tempo real, um dos principais objetivos deste projeto. Também é indicado neste capítulo, uma secção que menciona outras possíveis tecnologias a utilizar neste projeto.

3.1 *Frameworks Low Code*

Atualmente, os *websites* e as aplicações são desenvolvidas utilizando *frameworks low code*. Este conceito, permite que qualquer pessoa sem grande conhecimento em programação, possa criar o seu próprio *website*, porque apenas é necessário arrastar e soltar elementos para compor os *websites*/aplicações de forma mais intuitiva. A vantagem deste tipo de plataformas é a contribuição para a redução da quantidade de código, tempo e custos necessários para o desenvolvimento de um projeto. O desenvolvimento torna-se mais ágil e intuitivo e para além disso são plataformas escaláveis, podendo ser dimensionadas para grandes ou pequenas organizações.

3.1.1 WebCT

A WebCT é uma *framework* desenvolvida pela Altice Labs, recorrendo à *framework* Angular e à linguagem TypeScript.

Os componentes criados nesta *framework* cooperam entre si, que tem como objetivo automatizar o processo de desenvolvimento da construção de páginas *web* através de ficheiros de configuração que as representam. A estes ficheiros de configuração, são atribuídos os nomes de "*mocks*". Um *mock* é um ficheiro JavaScript Object Notation (JSON) (Figura 12), entendido como um esboço da página a ser construída. A figura seguinte, é uma representação de um ficheiro de configuração (*mock*).

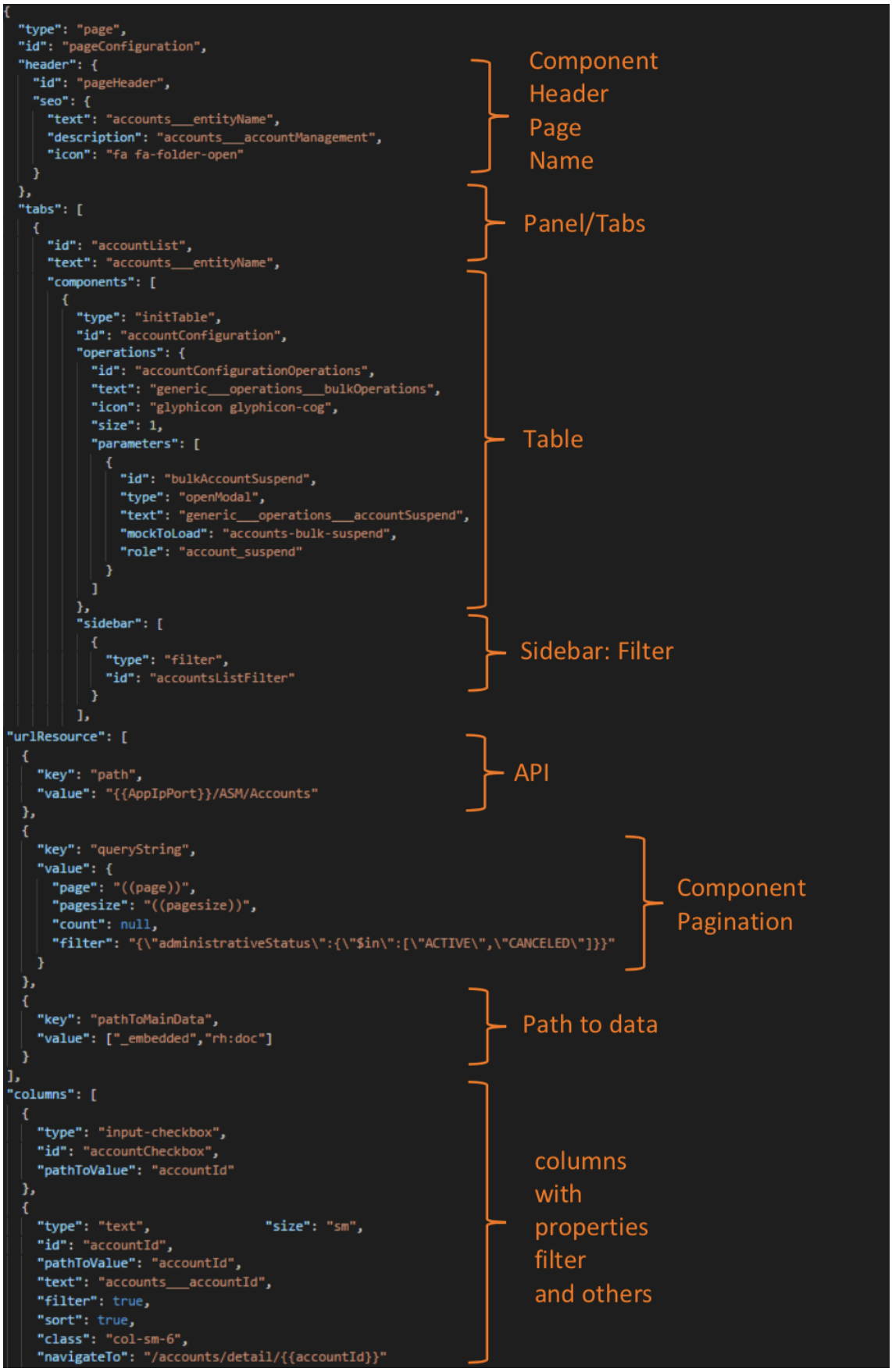

Figura 12 - Ficheiro de configuração (*mock*)

A WebCT, cobre as funcionalidades *end-to-end* de uma página *web* dinâmica tradicional ou mesmo de uma *progressive webapp* para responder a diversos dispositivos (*desktop* e móvel), desde a instanciação de componentes visuais, ligação dos mesmos, e integração com os respetivos *facades* (interfaces expostas dos serviços que tem de integrar) sendo moldável aos vários serviços e APIs que deve invocar.

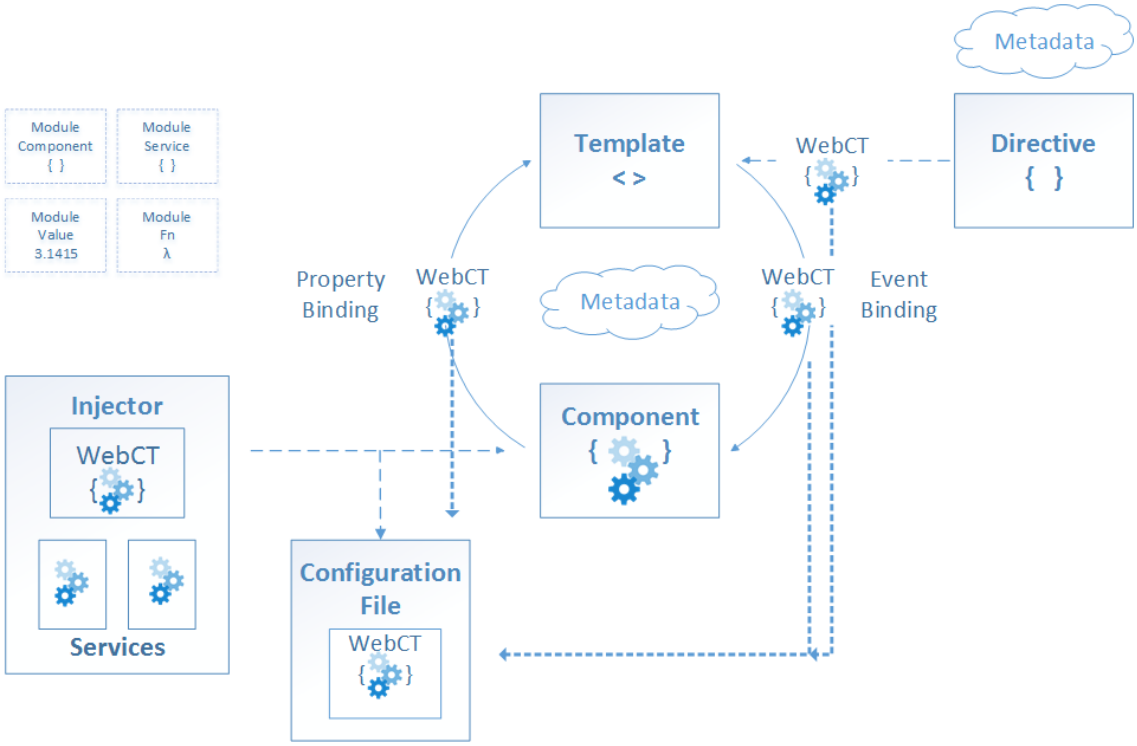

Figura 13 - Arquitetura WebCT

A *framework* WebCT é composta por três componentes principais: o *Robot*, a Palete e o *Custom*.

Através da Figura 13, é percetível que a WebCT, através do seu componente "*Robot*" é capaz de automatizar todas as interações da arquitetura Angular, através de ficheiros de configuração, que representam as diversas páginas da aplicação instanciando os componentes da sua "Palete" de componentes gráficos, com a responsabilidade de fazer a ligação entre os mesmos com a informação que invoca nos serviços HTTP. O componente *Robot* é ainda responsável por dinamizar todo o processo, recorrendo aos serviços HTTP para acesso às APIs que populam componentes como tabelas, *cards*, etc, fazem o *parser* dos ficheiros de configuração (*mocks),* que como mencionado anteriormente, são uma representação da página em ficheiro JSON, reutilizando assim os diversos componentes disponíveis no componente Palete (tabelas, *labels*, *cards*, botões, *headers*, etc.) (Figura 14). Por fim, o componente "*Custom*" representa os novos componentes com características especificas e especiais que são criados quando necessário e que ainda não existem, nem se encontram disponíveis no conjunto de componentes *web* da WebCT.

A figura seguinte, apresenta representações visuais de alguns componentes disponibilizados pela WebCT, desde tabelas, *cards*, *wizards*, *tabs*, filtros, etc.

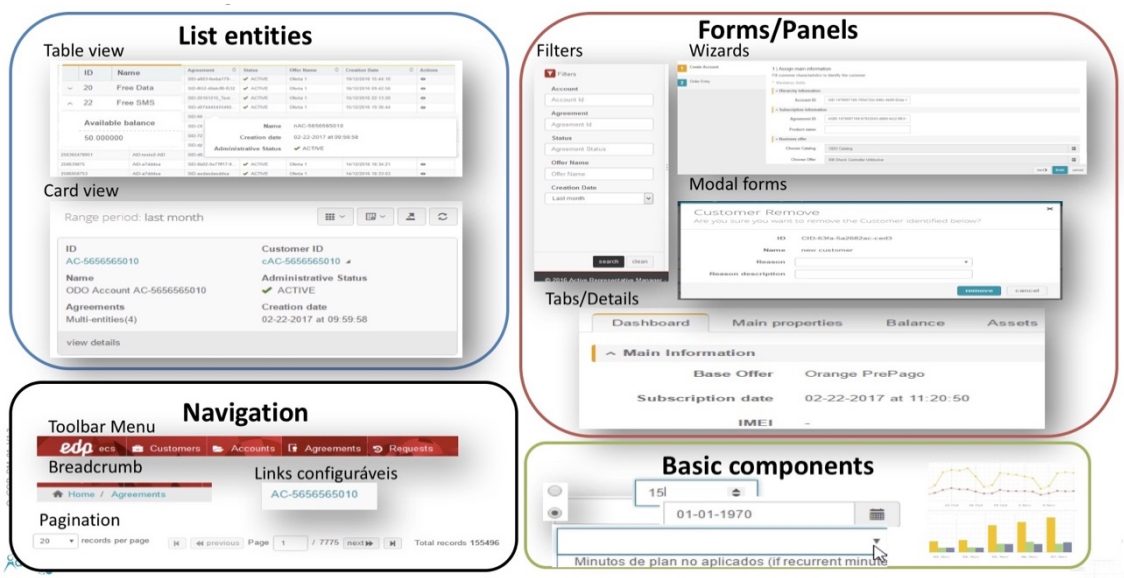

Figura 14 - Componentes WebCT

Algumas vantagens do WebCT:

- acelera o processo de desenvolvimento de aplicações *web*;
- permite reutilizar componentes *web*;
- novos recursos ou correção de *bugs* estão disponíveis para cada aplicação *web*;
- não é necessário entender, profundamente, sobre Angular para criar aplicações;
- criação de componentes personalizados estendendo um componente existente ou criação de um novo de raiz;
- componentes bem testados;
- *design* para arquiteturas de micro serviços;
- fácil integração com outros sistemas através de APIs;
- providencia uma grande colaboração nas aplicações *web*;
- móvel & *desktop*;
- lema: “Faça menos, ganhe mais!”;
- foco na produtividade.

A WebCT, está mais vocacionada para arquiteturas de micro serviços, uma vez que os componentes podem invocar diferentes serviços e dar uma informação consistente ao utilizador. Um exemplo, destas técnicas de criação de sistemas é o caso da Amazon, onde os produtos são fornecidos por um serviço, as imagens por outro serviço e os respetivos preços por outro serviço.

“O estilo da arquitetura de micro serviços é uma abordagem para o desenvolvimento de uma única aplicação como um conjunto de pequenos

serviços, cada um executado no seu próprio processo e comunicando com mecanismos leves como uma API de recursos HTTP" [49]. Cada serviço é desenvolvido em torno de um conjunto de regras de negócio específicas, e é implementado de forma independente.

Normalmente, as aplicações são construídas em três partes principais: uma interface de utilizador do lado do cliente, uma base de dados e uma aplicação do lado do servidor. A aplicação do lado do servidor trata dos pedidos HTTP, executa a lógica de domínio, recupera e atualiza os dados da base de dados e seleciona e preenche as *views* HTML para serem enviadas para o *browser*.

Uma aplicação com arquitetura de micro serviços permite desenvolver sistemas mais flexíveis, mais escaláveis e com manutenção mais simples do que as arquiteturas de sistemas monolíticas (construídos como uma unidade única) (Figura 15). Os serviços são independentemente implementáveis e escaláveis. Segundo Martin Fowler, um sistema considerado complexo que segue uma arquitetura de micro serviços tem a longo prazo, um custo de manutenção menor do que o de aplicações que seguem uma arquitetura monolítica [50].

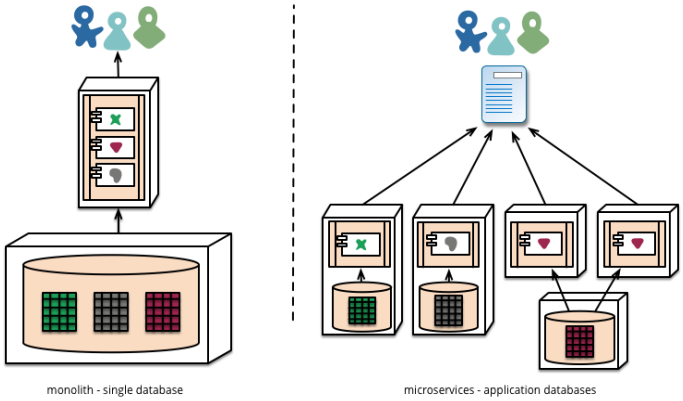

Figura 15 - Arquitetura Monolítica vs. Arquitetura de Micro Serviços [49]

Os micro serviços permitem criar um ecossistema de arquitetura que permite alterações a qualquer momento [51], permitindo a separação de responsabilidades, o que não é possível numa arquitetura monolítica. Empresas como Amazon, Netflix, eBay, Twitter, PayPal, etc. evoluíram da arquitetura monolítica para arquitetura de micro serviços [52].

Comparativamente com sistemas monolíticos, a arquitetura de micro serviços oferece diversas vantagens indispensáveis, tais como [53], [54]:

- autonomia e propriedade: cada equipa tem a capacidade de criar e implementar um micro serviço sem depender de outras equipas, oferecendo assim, mais autonomia e agilidade na criação dos projetos;

- independência e altamente escalável: equipas podem escalar partes da plataforma de maneira independente. Escalar com facilidade e rapidez à medida que vai sendo necessário;

- desacoplamento: os micro serviços são desacoplados, pelo que é possível fazer alterações num micro serviço de forma independente, sem ser necessário alterar o *software* todo;

- liberdade de tecnologia e linguagens: a flexibilidade dos micro serviços, oferece liberdade na escolha das tecnologias a utilizar, permitindo escolher a mais adequada para cada funcionalidade especifica. Permite misturar várias linguagens, *frameworks* e tecnologias de base de dados;

- flexibilidade: plataformas baseadas em micro serviços são mais robustas e económicas.

Tal como a WebCT, existem *frameworks low code* que permitem reutilizar código, tornando o processo automatizado e dinâmico. Dispõem de ferramentas gráficas WYSIWYG, "*What You See Is What You Get*", de *drag and drop* de componentes *web*. Algumas *frameworks* líderes de mercado são: OutSystems, Salesforce, Mendix, SkavaSTUDIO e Wix [55].

3.1.2 OutSystems

A OutSystems [56], é uma *framework* completa para desenvolvimento *low code* de aplicações *web* e móveis empresariais de forma rápida e flexível. Utiliza ferramentas de *drag and drop* e permite definir o modelo de dados, a lógica de negócio, os processos de fluxo de trabalho e as interfaces de utilizador [57]. Os componentes da aplicação são arrastados para uma superfície de trabalho, no entanto os programadores podem estender funcionalidade à aplicação e inserir os seus códigos personalizados em linguagens como C #, Java, SQL, HTML, CSS e JavaScript.

Ao contrário de algumas *frameworks* o objetivo da OutSystems é fazer com que os programadores continuem a manter o controlo na criação de interfaces, sendo que esta, apenas adiciona o que já é muito "intuitivo" para eles de modo a torná-los mais produtivos, promovendo assim a reutilização de código.

3.1.3 Salesforce

A Salesforce é uma plataforma *Customer Relationship Management* (CRM). As suas aplicações baseadas na *cloud* são direcionadas para vendas, serviços, *marketing* e não exige que sejam especialistas em IT que configurem ou façam a sua gestão [58]. Permite criar aplicações, arrastando componentes e automatiza processos de negócio [55].

3.1.4 Mendix

A Mendix enfatiza as capacidades de desenvolvimento colaborativo da sua plataforma, fornecendo a criação de modelos de dados, lógica e interfaces de utilizador e um editor de páginas WYSIWYG [59]. Oferece temas de *websites* pré-construídos, *layouts* de navegação, modelos de página e elementos de *design*. Assim como a OutSystems, oferece extensibilidade da aplicação para que programadores possam adicionar novas funcionalidades personalizadas.

3.1.5 SkavaSTUDIO

O SkavaStudio oferece uma interface WYSIWYG, que permite criar rapidamente aplicações sem necessidade de entender profundamente sobre programação. Procura maneiras de criar experiências mais ricas e contextualizadas para os seus clientes. Tradicionalmente, as equipas de *marketing* dependem das equipas de IT para procederem à criação dos *websites*, no entanto o SkavaStudio permite que as equipas de *marketing* possam criar os seus próprios *websites* ou aplicações e que as equipas de IT se foquem em projetos mais estratégicos [60].

3.1.6 Wix

A Wix é uma plataforma que permite que qualquer pessoa consiga criar a sua aplicação de forma simples e sem ser necessário que esta aplique "código". Permite criar uma aplicação utilizando *templates* e através de "*drag and drop*" de componentes [61]. No entanto também disponibiliza, o Wix Code, que é um ambiente de desenvolvimento integrado (IDE), para que programadores, consigam através de introdução de código, em linguagem JavaScript, personalizar ainda mais os *websites* e aplicarem as edições que necessitam [62].

3.2 Estudo de *Frameworks* Orientadas a Componentes *Web*

Uma *framework*, é uma ferramenta de desenvolvimento de *software* que promove a produtividade, a diminuição de tempo despendido em desenvolvimento e aplicações com mais qualidade [63]. Sempre que é necessário implementar elementos, é indispensável copiar código, por exemplo, HTML, CSS e JavaScript para aplicar no projeto. O ato de copiar código desencadeia grandes efeitos colaterais e por isso, é necessário muito cuidado para que o código não sofra conflitos externos e seja o mais flexível e reutilizável possível.

De forma a solucionar este problema, surgem os "Componentes *Web"*, um modelo, que como o nome sugere tem a capacidade de "compor" a aplicação *web* em pequenas partes reutilizáveis e modulares que se encaixam nas

aplicações *web* como peças de um puzzle [64], oferecendo a flexibilidade necessária para lidar com os requisitos extremamente dinâmicos das aplicações *web* [65].

Os componentes *web* consistem num grupo de cinco especificações: *templates*, *decorators*, *custom elements*, *shadow* DOM e *imports* [66].

- ***Templates***: definem excertos de código que são totalmente inertes à página, mas podem ser ativados para utilização posterior através do respetivo ficheiro JavaScript.

- ***Decorators***: aplicam os *templates* com base em ficheiros CSS para criar alterações visuais e comportamentais aos documentos.

- ***Custom Elements***: são elementos personalizados, que permitem que os autores definam os seus próprios elementos, com novos nomes e novas interfaces de *scripts*.

- ***Shadow DOM***: local onde parte do elemento é encapsulada e "escondida" pelo *browser*, para uma composição mais confiável dos elementos.

- ***Imports***: definem quais *templates*, *decorators,* e *custom elements* podem ser acedidos.

Em suma, as *frameworks web* da nova geração são orientadas a componentes *web*, de modo a resolver melhor o problema da reutilização de componentes vs. código. Seguidamente, são descritas algumas *frameworks*/bibliotecas como React.js, Angular e Vue.js que têm como objetivo agilizar e facilitar o processo de criação de componentes *web* [67].

3.2.1 React.js

A React.js, foi desenvolvida em 2013, pela equipa do Instagram/Facebook e é uma biblioteca JavaScript, *open-source*, ideal para desenvolver interfaces de utilizador [68].

Visto que as aplicações *web* não são estáticas, o DOM (*Document Object Model*) necessita de ser alterado sempre que ocorre um evento (eventos por parte do utilizador ou de um servidor). Quando ocorrem estes eventos, os dados que representam um estado do modelo de dados são atualizados e quando um estado se altera, o objetivo é aplicar essa alteração também no estado da interface do utilizador [69]. Este e o estado do modelo de dados devem estar sincronizados entre si, a sincronização entre ambos é realizada através de *two-way data binding* (sincronização bidirecional). No entanto, a React utiliza outra forma para realizar a sincronização de ambos os estados, chamado *Virtual DOM (V-DOM).*

O *V-DOM* é a grande mais valia da React. Dado que a manipulação do DOM é algo complexo, o *V-DOM* (representação em JavaScript do DOM real) é o objeto manipulado em vez do DOM real. Quando existe uma alteração no modelo de dados, em vez de se atualizar o DOM real, é gerada uma representação do *V-DOM*, enviando o novo estado do modelo como parâmetro e é utilizado um mecanismo de comparação, para obter as diferenças entre a representação do *V-DOM* e do DOM real. O DOM real é atualizado conforme as diferenças observadas [69].

Por outro lado, a React não é capaz de criar um projeto *Single Page Application* (SPA), que é uma forma de desenvolver uma aplicação completa numa única página *web*, necessita de um ecossistema de bibliotecas para auxiliar este processo [67]. No ecossistema da React, uma aplicação é composta por outras bibliotecas especializadas, tais como, *react-router* e *react-redux* e com estas, já é possível criar um projeto SPA complexo [67] utilizando React.

Esta biblioteca, foca-se na utilização de JavaScript ES6 e JavaScript XML (JSX) [70], sendo JSX um tipo de extensão XML para a especificação ECMAScript [71]. É utilizada por várias entidades, tais como: Airbnb, Uber, Netflix, Twitter, Pinterest, Reddit, Wix, Tumblr entre outras [69].

3.2.2 Vue.js

A Vue, é uma *framework* para construção de interfaces *web* interativas e foi desenvolvida por um membro original da equipa da AngularJS, em 2014. A *framework* AngularJS corresponde à primeira versão da *framework* Angular [72].

Tecnicamente, a Vue.js baseia-se no padrão *Model View View Model* (MVVM) que conecta a *view* e o modelo através de *data bindings* [73], sendo o *View Model* responsável pela sincronização entre ambos [74]. Este padrão é muito semelhante ao padrão de *design* mais influente e conhecido, o *Model View Controller* (MVC), que separa as aplicações em camadas, de modo a permitir uma fácil perceção e modificação independente de cada unidade especifica. A camada de interação com o utilizador (*view*), a camada de manipulação dos dados (*model*) e a camada de controle (*controller*) que representa a interação com o utilizador [75].

A Vue utiliza JavaScript ES5 ou ES6 [76] e é fácil de integrar com outras bibliotecas ou projetos, necessitando apenas também de um ecossistema para ajudar a criar vários tipos de projetos. As bibliotecas *vue-router* e *vuex* são indispensáveis para ajudar a criar projetos SPA [67]. A partir da versão 2.0, Vue.js passa a suportar *Virtual-DOM*, que veio melhorar o seu desempenho relativamente à versão 1.0.

3.2.3 Angular

A Angular é uma *framework* que permite desenvolver aplicações *web* e móveis. Esta *framework* surgiu de uma reformulação da *framework* já existente, AngularJS, sofrendo alterações ao nível de código, filosofia e linguagem [67]. No entanto, a AngularJS continua disponível para utilização.

Ao nível da linguagem, as aplicações Angular podem ser desenvolvidas em TypeScript, que possui um foco muito forte na utilização de programação orientada a objetos, tornando a *framework* mais apelativa para programadores de *backend* [65]. No entanto, no momento da compilação, o compilador compila os programas e emite para linguagem JavaScript, para que estes possam ser executados em diferentes ambientes de execução, visto que os *browsers* não "percebem" TypeScript [77].

A linguagem TypeScript, oferece vantagens como *static typing*, que permite a deteção de *bugs* à medida que os programadores escrevem *scripts*, sendo uma melhor opção para projetos de grandes dimensões, porque o *refactoring* com ferramentas TypeScript é mais fácil e mais rápido, oferece melhor colaboração através de *type safety*, que é uma funcionalidade que deteta erros durante o desenvolvimento de código e para além disso, o *auto complete* e o *dynamic type* podem ajudar a aumentar a produtividade dos programadores [78].

Ao contrário da biblioteca React, que utiliza o *V-DOM* para sincronização dos dados com a interface de utilizador, a *framework* Angular recorre a *two-way data binding* para o mesmo efeito que consiste numa maneira simples de manter a *view* e o modelo em sincronia. Se a *view* ou o modelo alteram, ambos são imediatamente sincronizados [79].

Esta *framework* que permite a criação de aplicações SPA, necessita de uma forma para "trocar de página". É possível criar um sistema de rotas, que quando solicitadas atualizam apenas partes da página, mantendo sempre a mesma base [80]. Ou seja, em aplicações SPA, apenas parte da *view* necessita ser alterada, diminuindo assim o tráfego de requisições.

A Angular é utilizada por várias entidades, como: Google (o seu criador), Wix, weather.com, healthcare.gov e Forbes [69].

3.2.4 Comparação

De modo a clarificar melhor as comparações entre as *frameworks* (Angular e Vue) e a biblioteca (React.js) foram analisados os seguintes tópicos de comparação por parte da comunidade de programadores [76]:

- Gestão de Estado e *Data Binding*

Uma grande diferença entre a React e a Angular é a utilização de *one-way* vs. *two-way binding*. Na *framework* Angular, o estado do modelo é alterado quando o elemento da UI é atualizado, na React o modelo é atualizado primeiro e posteriormente é feito o *render* do elemento da UI. Vue.js oferece suporte para ambas as ligações, *one-way-binding* e *two-way-binding*.

- Padrões de Design

Angular é baseada no padrão MVC, enquanto que React apenas tem a parte "*view*" do modelo e a Vue foca-se na parte *View Model*.

- *Size* & Performance

A React e a Vue utilizam *V-DOM*, que é suposto melhorar o desempenho. Vue é conhecida por superar a Angular e a React, no entanto a Angular pode oferecer melhor desempenho em aplicações de grande escala [70].

- Curva de Aprendizagem

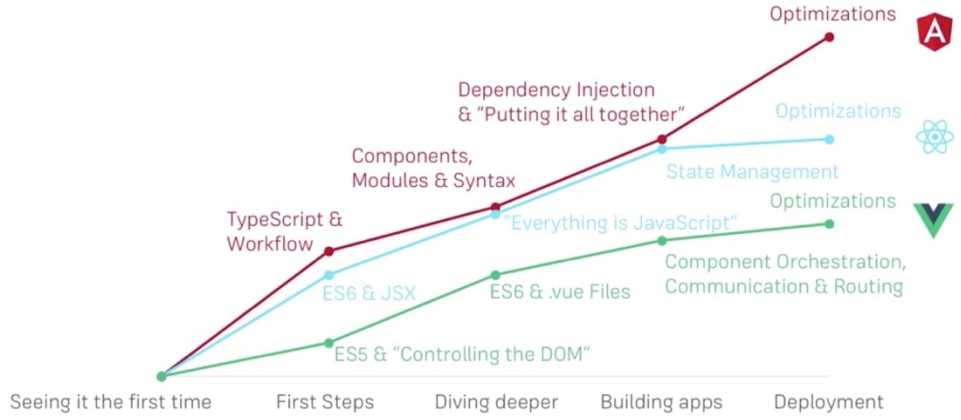

Figura 16 - Curvas de Aprendizagem Angular, React e Vue [81]

Segundo a Figura 16, a *framework* Angular apresenta a curva de aprendizagem mais íngreme, uma vez que requer a aprendizagem de uma nova linguagem (TypeScript), diferente do JavaScript habitual. Permite injeção de dependência (ou seja, é possível injetar um serviço num componente, para que esse componente tenha acesso a esse serviço [82]) e otimizar uma aplicação requer um pouco de trabalho, assim como obter benefício de funcionalidades como *lazy loading*, que permite que um módulo seja apenas carregado quando é invocado numa rota, reduzindo a carga e memória necessária no *browser* para carregar todos os módulos de início, ou alguns que possam vir a não ser

utilizados [83] ou *ahead-of-time compilation*, que compila o código HTML Angular e TypeScript em código JavaScript [84].

A biblioteca React também apresenta uma curva de aprendizagem bastante íngreme, embora utilize JavaScript, usa a versão ES6. Quando é utilizada esta versão, é necessário um fluxo de trabalho e de um ambiente de desenvolvimento que permita escrever código em ES6 e compilá-lo para ES5 para que execute também em *browsers* mais antigos e criados antes da sua criação. Isto representa um desafio adicional, especialmente para programadores com pouca experiência. React utiliza JSX, uma sintaxe que mistura HTML, mas ainda é JavaScript, o que significa que a forma como se manipula o código HTML pode ser difícil de entender inicialmente. A gestão de estado (consiste em todos os componentes React terem um estado e cada vez que esse estado é alterado, é aplicado *rendering* ao módulo) é outro tópico que se pode tornar problemático, devido a alguma complexidade. Otimizar uma aplicação que utiliza React não é tão complexo quanto otimizar uma aplicação que utiliza Angular, visto que para uma aplicação criada com Angular obter um elevado nível de otimização é crucial aproveitar os benefícios de funcionalidades como *lazy loading*, utilizando corretamente *ahead-of-time compilation*. Implementar estas funcionalidades pode ser um desafio.

Entre as três, a Vue.js é a que apresenta uma curva de aprendizagem mais baixa. Funciona bem com ES5 e ES6. Separa o *template* do código (JavaScript) assim como a Angular. Ao utilizar componentes e diretivas, é fácil e intuitivo manipular o *template* e exibir os seus dados. O nível de complexidade aumenta quando o objetivo é desenvolver aplicações mais complexas. É necessário aprender sobre *workflows* utilizando ficheiros .vue, que basicamente são ficheiros que tornam a criação de componentes mais fácil no final do desenvolvimento. Também é necessário entender como criar componentes, conectá-los uns aos outros e como gerir o estado da aplicação.

Em suma, a escolha entre React, Angular e Vue depende de quais características agradam a cada programador e quais são as mais adequadas para o projeto em questão ou para as tecnologias já em vigor. Todas as três *frameworks* e bibliotecas são excecionais com experiência e firmeza suficientes no mercado. Em seguida, está representada uma tabela síntese das características da React, Vue e Angular (Tabela 4).

Biblioteca/Framework	React	Vue	Angular
Estável	Sim	Sim	Sim
Suportado por uma grande comunidade	Sim, comunidade enorme suportada pelo apoio do Facebook	Não é enorme, mas é grande o suficiente e é suportada pela Laravel e Alibaba	Sim, comunidade enorme suportada pelo apoio do Google
Boa documentação	Sim	Sim	Sim
Fácil de aprender	Relativo	Sim	Requer aprendizagem de TypeScript
Baseada em componentes	Sim	Sim	Sim
Framework SPA	Não	Não	Sim
Bom desempenho em aplicações de grande escala	Sim	Não	Sim

Tabela 4 - Tabela Comparativa entre React, Vue e Angular

A Tabela 4, é referente à síntese de características da Angular, da React e da Vue. Todas apresentam uma forte estabilidade no mercado, suportadas por enormes comunidades, sendo que a comunidade da Vue, não é enorme, mas é grande o suficiente. Todas, apresentam boa documentação e são orientadas a componentes. Relativamente à facilidade de aprendizagem, a Vue acaba por apresentar uma curva de aprendizagem mais fácil, a aprendizagem de React é relativa, pois a linguagem utilizada é o conhecido JavaScript e Angular, acaba por requerer a aprendizagem de uma nova linguagem, o TypeScript. A Angular, permite a criação de aplicações SPA, enquanto que a React e a Vue necessitam de integração com outras bibliotecas para "conseguirem" criar aplicações SPA. Por fim, a Angular e a React são capazes de obter melhor desempenho em aplicações de grande escala, comparativamente com a Vue.

3.3 Estudo aprofundado sobre Angular

Tendo em conta que a WebCT é uma *framework* desenvolvida sobre Angular foi realizado um estudo mais aprofundado sobre esta *framework.* A Angular é composta pela Angular CLI, Angular SPA (*cliente-side rendering*) e Angular Universal (*server-side rendering*) sendo que atualmente a *framework* WebCT utiliza apenas Angular CLI e SPA, no futuro, passará também a utilizar Angular Universal.

Angular CLI é uma interface de linha de comandos cujo objetivo é facilitar a criação e gestão das aplicações. Permite gerar componentes, rotas, serviços,

filtros, etc, através de um simples comando [85] e baseia-se no Webpack, uma ferramenta que ajuda a processar e agrupar ficheiros como TypeScript, JavaScript, CSS, HTML e imagens [86]. Na Angular SPA, quando é realizado o primeiro pedido de página do cliente para o servidor, o servidor envia em resposta toda a aplicação para o lado do cliente. As seguintes interações entre o cliente e servidor é só para troca de dados. Na Angular Universal, quando é realizado o primeiro pedido de página do cliente para o servidor, o servidor envia em resposta os ficheiros (HTML, CSS, JavaScript/TypeScript, etc.) relativos apenas à página pedida para o lado do cliente.

3.3.1 Arquitetura

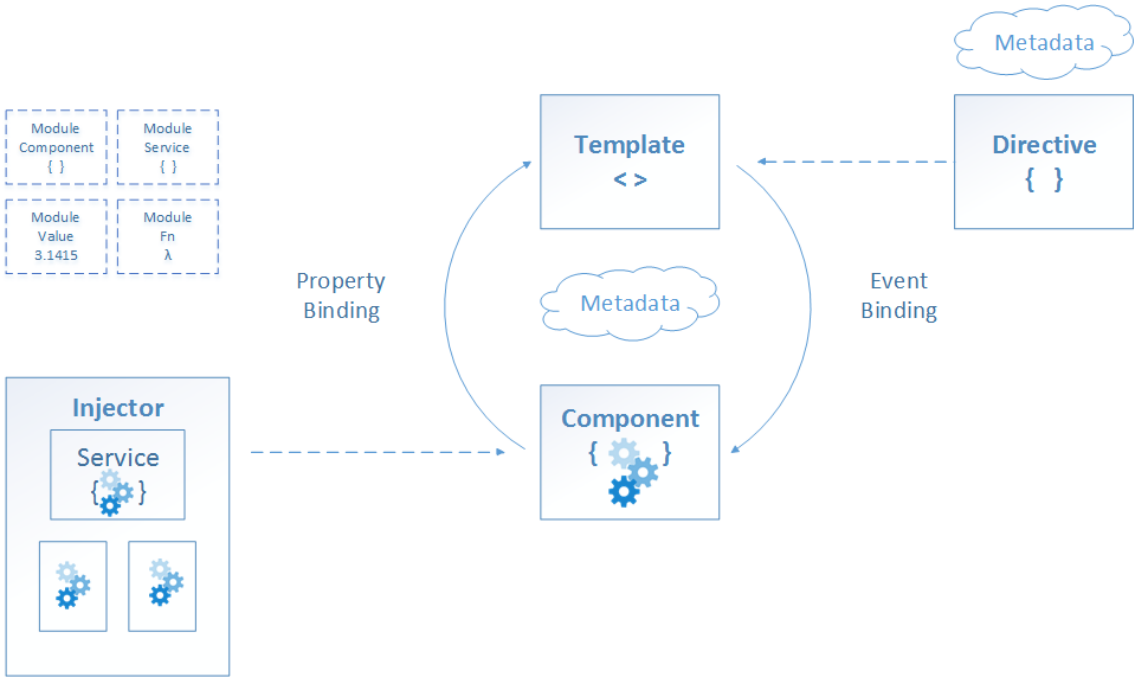

Figura 17 - Arquitetura Angular [87]

Tudo na Angular são componentes. A arquitetura da *framework* Angular contém os seguintes módulos: módulo, componente, *template*, metadados, *data binding*, serviço, diretiva e injeção de dependência (Figura 17).

As aplicações Angular seguem uma estrutura modular, visto que uma aplicação é construída a partir de vários módulos e cada **módulo** é dedicado a um único propósito. Ou seja, o módulo consiste num bloco de código que pode ser utilizado para executar uma única tarefa.

Um **componente** é uma classe *controller* que define o comportamento da *view* que se encontra no ***template*** e a comunicação/coordenação entre ambos é realizada através de *data binding* (Figura 18), onde ***property binding*** é a comunicação no sentido componente para o *template* e ***event binding*** é a comunicação no sentido do *template* para o componente. No *one-way data binding* a vinculação de dados que segue o sentido do componente para o

template pode ser realizado utilizando a *Interpolation* {{valor}} , *Property Binding* `[propriedade]="valor"` e a vinculação que segue o sentido do *template* para o componente pode ser realizada através do *Event Binding* `(evento)="handler"`.

Existe ainda o *two-way data binding* que funde a vinculação de propriedades e eventos numa única notação utilizando a diretiva ngModel `<input>[(ngModel)]= "cliente.nome">`. Obtém-se assim, uma sincronização bidirecional e sempre que o valor for alterado, no componente ou no *template*, através da sincronização, este é atualizado em ambos os lados.

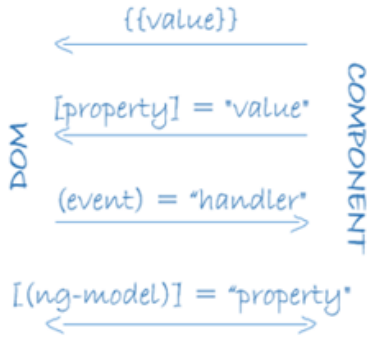

Figura 18 - Data binding [88]

Os **serviços** servem para isolar uma lógica de negócio específica para um domínio que pode ser injetado através da **injeção de dependência** para poder ser utilizado por componentes. Geralmente, o serviço é uma classe que pode realizar algo específico, como serviço de registo, serviço de dados, serviço de mensagens, a configuração da aplicação, etc.

A **diretiva** é algo que podemos complementar a um *template*. É uma classe que contém metadados que será anexada à classe pela anotação @Directive.

Existem três tipos de diretivas

- *Component Directive*: cria um *controller* personalizado utilizando *view*, *controller* e elemento HTML;
- *Decorator Directive*: decora os elementos usando um comportamento adicional;
- *Template Directive*: converte HTML num modelo reutilizável.

Existem dois tipos de arquiteturas

- Estrutural, alteram o *layout* adicionando, removendo e substituindo elementos no DOM. ex: ngFor e ngIf;

- Atributo, alteram a aparência ou o comportamento de um elemento existente. Ex: ngModel.

Por fim, os **metadados** indicam como processar uma classe, sendo necessário adicionar metadados ao código para a Angular saber o que fazer. Os metadados presentes no `@Component` indicam à Angular onde arranjar os blocos principais de construção especificados para o componente. O *template* juntamente com os metadados e o componente descrevem uma *view*.

3.3.2 Node.js, NPM e Yarn

A Angular executa num ambiente gerado pelo Node.js [89], que é baseado num modelo assíncrono de eventos I/O [90]. O Node Package Manager (NPM) [91] e Yarn [92] são gestores de *packages* e dependências, que consistem num repositório para publicação e reutilização de projetos *open-source*.

Node.js é um ambiente cliente-servidor JavaScript. Contrariamente à maioria dos outros ambientes modernos, um processo Node não depende de *multithreading* para suportar a execução simultânea da lógica de negócios, porque é baseado num modelo assíncrono de eventos I / O. Outra vantagem do Node é que não permite o bloqueio de um processo, porque não há bloqueios e quase nenhuma função executa diretamente I / O, por isso o processo nunca bloqueia [90]. O Node.js oferece bom desempenho, utiliza JavaScript que suporta *callbacks* de eventos e uma grande quantidade de *packages*/bibliotecas disponíveis para a realização de várias tarefas comuns (acesso à base de dados, *log*, servidores *web*, etc.).

O *Node Package Manager* (NPM) é um gestor de *packages* do Node.js, que consiste num repositório online para publicação de projetos *open-source*, tornando mais fácil a partilha de código entre os programadores. Estes "pedaços" de código reutilizáveis são chamados de *packages* ou módulos. Um *package* é apenas um diretório com um ou mais ficheiros, juntamente com um ficheiro chamado "package.json" que contém metadados sobre o *package* [91].

O NPM é composto por três partes distintas: o *website*, o registo e a interface de linha de comandos. O *website* permite que os utilizadores encontrem os *packages*, o registo é a base de dados de informações sobre os *packages* e a interface de linha de comandos permite que os programadores publiquem os *packages* no registo ou realizem *download* dos *packages* que desejam instalar [91].

Outra vantagem na utilização do NPM é que este é um gestor de dependências, que declara e resolve dependências [93]. Uma dependência é utilizada para especificar um módulo que seja necessário para executar outro módulo (representado pelo package.json) [94]. Quando existe um projeto Node com um ficheiro package.json, executa-se o comando "npm install" na pasta raiz

do projeto e o NPM instala todas as dependências listadas no ficheiro package.json, simplificando a forma de instalar projetos Node de repositórios Git. O ficheiro package.json é o ponto de partida para qualquer projeto NodeJS e é responsável por descrever o projeto, informar sobre a versão do Node e do NPM, *url* do repositório, versão do projeto, dependências de produção e de desenvolvimento entre outros [94].

O Yarn também é um gestor de *packages* como o NPM que permite que programadores partilhem código entre si de forma rápida, segura e confiável. Foi lançado em 2016, pelo Facebook [92]. Uma diferença entre o NPM e o Yarn, é que o Yarn armazena em cache todos os *downloads* de *packages*, e assim nunca necessita de fazer novamente *download* dos mesmos. Permitindo que seja possível instalar um *package* em modo *offline*, caso este tenha sido instalado anteriormente. Para além disso, o Yarn utiliza *lockfiles* e um algoritmo de instalação que é determinístico para operações de instalação. Ou seja, as mesmas dependências são instaladas da mesma maneira em qualquer máquina, independentemente da ordem de instalação. Os *lockfiles* bloqueiam as dependências numa versão específica. Garantindo que qualquer instalação que funcione, funciona exatamente da mesma forma noutro sistema [95].

3.4 Tecnologias e Sistemas de Apoio

Para o desenvolvimento deste sistema, foi realizado um estudo sobre possíveis tecnologias e *frameworks* a integrar neste sistema.

- **MongoDB**

O MongoDB é uma base de dados não relacional, *open source,* orientada a documentos, ou seja, as bases de dados são compostas por coleções de documentos. A principal diferença perante as bases de dados relacionais é que em vez de os dados serem armazenados numa estrutura rígida (ex: tabelas) são armazenados em documentos.

Nos documentos, os dados têm uma estrutura chave/valor (objetos JSON) (Figura 19), sendo possível acrescentar novos atributos a documentos de forma individual sem que os restantes sofram alterações.

Figura 19 - Coleção de Documentos MongoDB [45]

- **RESTHeart**

"O RESTHeart é utilizado como um *middleware* para expor os documentos do MongoDB através de uma API HTTP (através dos métodos GET, POST, PUT, PATCH, DELETE), no qual oferece um modelo compatível com uma representação *Hypermedia as the Engine of Application State* (HATEOAS)" [96].

Segundo Roy Fielding [97], *Representational State Transfer* (REST) é uma imagem do *design* da aplicação e de como esta se comportará: uma rede de *websites* onde o utilizador progride numa aplicação selecionando as ligações (transições de estado), que tem como resultado a página seguinte que representa o estado seguinte da aplicação que esta a ser apresentada ao utilizador.

A existência de recursos (elementos de informação) é um conceito muito importante no REST, estes podem ser usados utilizando um identificador global (URI) para manipular estes recursos, os componentes da rede (clientes e servidores) comunicam através de uma interface padrão (HTTP) e trocam representações de recursos (os ficheiros são recebidos e enviados).

Uma aplicação pode interagir com um recurso conhecendo o seu identificador e a ação solicitada, não necessita saber se existem caches, *proxys* entre esta e o servidor (guarda a informação). A aplicação deve compreender o formato da informação de retorno (a representação), que geralmente é um documento em formato HTML ou XML.

O princípio HATEOAS é uma das principais propriedades do REST e consiste num conjunto coordenado de restrições de arquitetura, onde o hipertexto indica o que é permitido ou não. Ou seja, representa uma fronteira a nível das navegações. Por exemplo, a aplicação prática do HATEOAS é o protocolo HTTP, onde vários servidores podem comunicar com vários clientes, e ambos podem evoluir de forma independente uns dos outros (*low coupling*).

A Figura 20, representa o "*Richardson Maturity Model*", que é um modelo de maturidade RESTful desenvolvido por Leonard Richardson que divide os principais elementos de uma aproximação REST em três níveis [98].

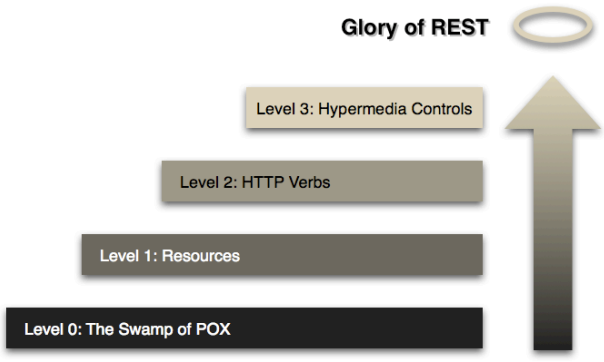

Figura 20 - Richardson *Maturity Model* [50]

Nível 0: o ponto de partida para o modelo é a utilização de HTTP como um sistema de transporte para interações remotas, mas sem usar nenhum dos mecanismos da *web*. POX é um XML simples utilizado para as comunicações.

Nível 1: introdução de recursos para abordar a questão de lidar com complexidade aplicando "dividir e conquistar", dividindo um *endpoint* grande de serviço em múltiplos recursos. Para identificar os recursos é utilizada uma sintaxe universal.

Nível 2: introduz um conjunto padrão de métodos HTTP com o objetivo de tratar situações similares da mesma forma.

Nível 3: utilização de hipermédia, introduzindo explorabilidade, promovendo uma maneira de ter um protocolo mais auto documentado.

Dentro do núcleo HATEOAS encontra-se o conceito de hipermédia (utilizado no nível 3 do modelo) ou por outras palavras: *links*. O fato de o servidor oferecer um conjunto de *links* para o cliente, permite que este mude a aplicação de um estado para outro, através de um *link*. Os dados e o *link* representam o estado do sistema, ou do recurso dentro do sistema. O estado do sistema é importante pois é preciso saber onde se está e para onde se pode ir.

O servidor não precisa registar que cliente é que está a aceder, nem qual o estado deste porque o próprio cliente sabe que as únicas transições possíveis são aquelas para as quais existe um *link* disponível. O cliente não precisa implementar nenhuma lógica para descobrir quais ações são possíveis baseadas no estado atual do recurso porque os próprios *links* já dizem o que pode ou não ser feito.

Muitos serviços exigem que o cliente envie informações específicas do utilizador (por exemplo, id) em cada URI de solicitação. Os mesmos pedidos de dois clientes diferentes parecem exclusivos das caches da *web*, porque estas utilizam URIs como chaves para os dados. O envio de informações específicas do utilizador, muitas vezes é desnecessário, mas é amplamente utilizado pelos *providers* de serviços *web* para limitar o número de acessos de cada cliente. Isto também afeta a capacidade da cache [99].

4 Solução Proposta e Desenvolvimento

Este capítulo foca-se na escolha das tecnologias a serem utilizadas na solução e em detalhes da implementação da arquitetura desenvolvida. Numa primeira fase, é indicada a solução proposta, assim como as tecnologias propostas para a mesma. Posteriormente, é apresentada a arquitetura e os diagramas de sequência que ajudam a entender o fluxo do sistema, assim como as suas funcionalidades. Por fim, será descrita em pormenor cada bloco da arquitetura da solução.

4.1 Introdução

Tendo em conta os principais objetivos, a solução proposta passa pela escolha da utilização de tecnologias flexíveis, modulares, orientadas a componentes que permitam facilmente criar interfaces dinamicamente e de tecnologias de ML para detetar padrões de comportamento nas interações que cada utilizador tem com a interface, de modo a que o sistema seja capaz de recomendar e oferecer a interface mais adaptável e personalizável a cada utilizador.

De forma a implementar a solução proposta, foram utilizadas as seguintes tecnologias:

- **WebCT e Angular**

Na Altice Labs, a WebCT foi criada num paradigma da arquitetura de micro serviços, surgindo da necessidade de desenvolver *widgets* adaptados às necessidades das aplicações/portais que suportam. Devido à sua flexibilidade, a WebCT e por consequência a Angular foram, respetivamente, a *framework low code* e a *framework* orientada a componentes *web* utilizadas neste projeto.

- **Node e NPM**

O Node.js foi o ambiente cliente-servidor escolhido, por ser baseado num modelo assíncrono de eventos, impedindo o bloqueio de processos. Outro fator relevante nesta escolha recai no fato de a Angular também utilizar Node.js.

Relativamente ao gestor de *packages*, o NPM foi o gestor escolhido, tendo em conta que já é instalado automaticamente com o Node.js.

- **ML e Tensorflow**

De forma, a detetar padrões de comportamento nas interações que cada utilizador tem com a interface e a identificar comportamentos de novos utilizadores no portal com utilizadores que já tem padrões identificados, a

escolha recaiu na utilização de técnicas de ML para o efeito. A TensorFlow foi a *framework* de ML escolhida para utilizar neste sistema, devido à sua comunidade, assim como à boa documentação, facilidade de uso, flexibilidade e por permitir fácil integração com computação distribuída.

- **MongoDB**

Devido à sua estrutura flexível, o MongoDB foi a base de dados escolhida para integrar este sistema. Apesar de a estrutura dos dados, atualmente, estar bem definida, devido à continuação da evolução deste projeto, é previsível que a estrutura de dados sofra alterações a curto prazo.

- **RESTHeart**

No contexto deste projeto, é utilizado RESTHeart para fazer as comunicações de dados no sistema.

4.2 Arquitetura

Na imagem seguinte, é possível verificar o diagrama de arquitetura completo.

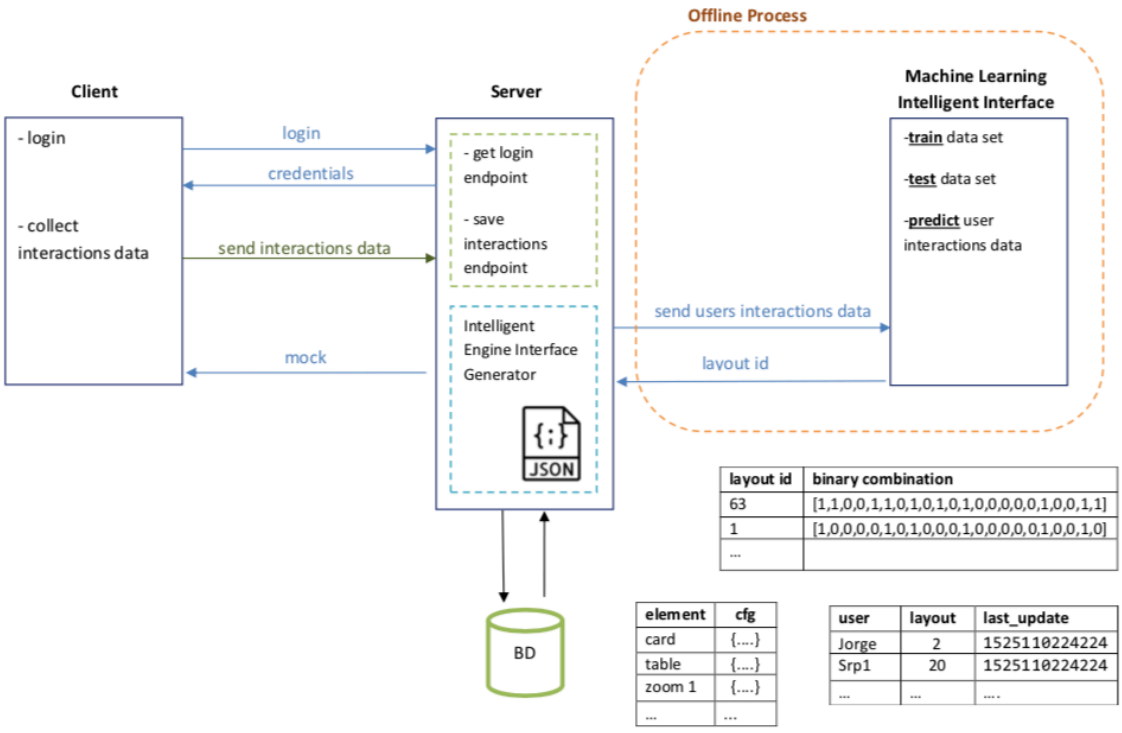

Figura 21 - Arquitetura do Sistema

A arquitetura deste sistema divide-se em três blocos principais. O primeiro bloco é a “Recolha de Dados”, responsável por recolher dados de informações

relativos às interações que cada utilizador tem com a interface. O segundo bloco, "*Machine Learning Intelligent Interface*" é responsável por aplicar um modelo de ML aos dados recolhidos no primeiro bloco de modo a obter o id do *layout* mais adequado a cada utilizador. Por fim, o terceiro bloco, "*Intelligent Engine Interface Generator*" tem como função gerar um *mock* (ficheiro .json) personalizado para cada utilizador que quando efetua *login*, já tem um *layout* atribuído. O primeiro bloco, encontra-se implementado em TypeScript e Angular. O servidor em Node.js e o segundo bloco (processo *offline*), "*Machine Learning Intelligent Interface*" em Python.

Seguidamente, são apresentados os diagramas de sequência, que descrevem o fluxo do sistema:

1) *Login* de utilizador sem *layout* atribuído

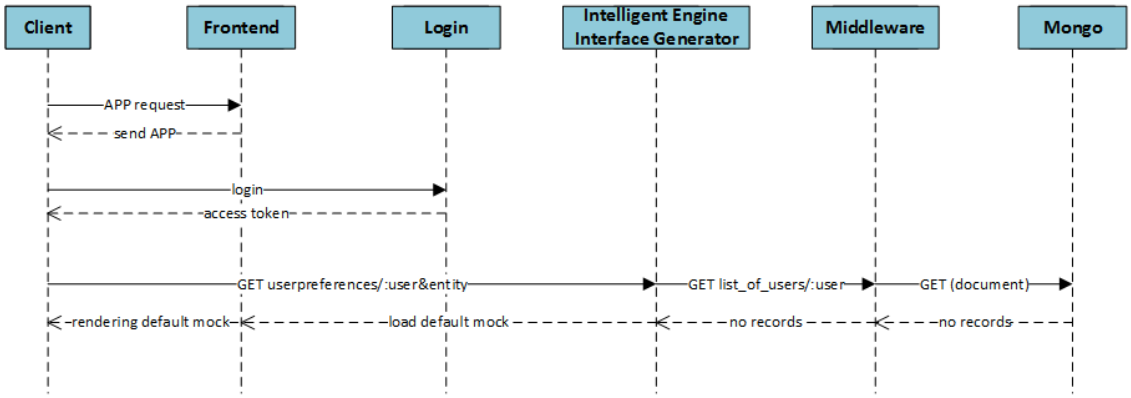

Figura 22 - Diagrama de sequência caso de uso *login* de utilizador sem *layout* atribuído

Como é possível observar, na Figura 22, primeiramente é efetuado o pedido da aplicação ao servidor na parte do *frontend* por parte do cliente. Seguidamente é devolvida a aplicação face a esse pedido, permitindo que o utilizador efetue o *login* na aplicação, sendo posteriormente enviado um *token* de acesso.

Quando o utilizador seleciona uma entidade disponível na aplicação, é realizado um pedido GET para o serviço "*Intelligent Engine Interface Generator*" para obter as preferências *layout* do utilizador para a entidade selecionada. Por sua vez, o serviço necessita realizar um pedido GET através de *endpoint* a um *middleware* para aceder à lista de utilizadores e respetivos ids *layouts*. Cada utilizador pode ter no máximo um *layout* para cada entidade da aplicação. O *middleware* efetua o pedido GET do documento Mongo correspondente e não é retornado nenhum documento porque este utilizador ainda não tem interações com a interface. O *middleware* não envia nenhum registo para o "*Intelligent Engine Interface Generator*", que interpreta e envia o *load* do *mock default* para a aplicação *frontend*, que por sua vez, efetua o *rendering* do *mock default.*

2) Recolha de dados

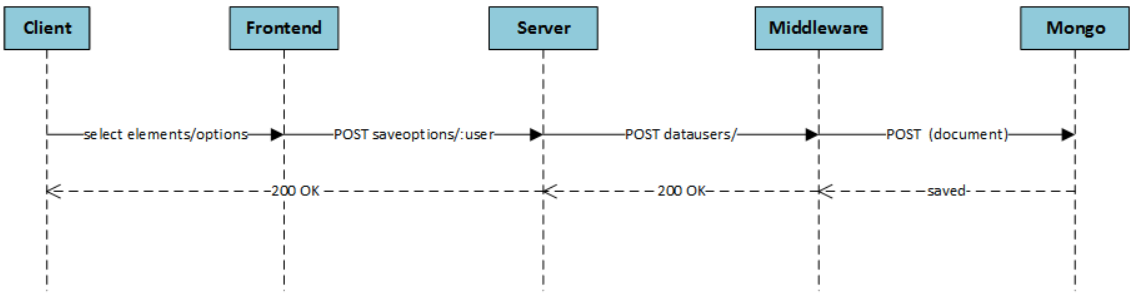

Figura 23 - Diagrama de sequência caso de uso recolha de dados de *input*

Na Figura 23, é possível verificar que, na parte do cliente, os utilizadores podem interagir com a aplicação, selecionando componentes ou configurações dos mesmos, sendo recolhidos na aplicação *frontend* os dados resultantes dessas interações. Seguidamente, para cada interação recolhida, a aplicação *frontend* efetua um pedido POST para o servidor com o objetivo de guardar os dados desta interação, assim como o *username* do utilizador que realizou estas interações. O servidor, por sua vez, efetua um pedido POST para o *middleware* para guardar os dados na coleção "data_users" que contem os dados relativos a todas as interações de todos os utilizadores. Em seguida, o *middleware*, efetua um pedido POST para adicionar um novo documento à coleção com os dados desta interação.

3) Processo *Offline Machine Learning Intelligent Interface*

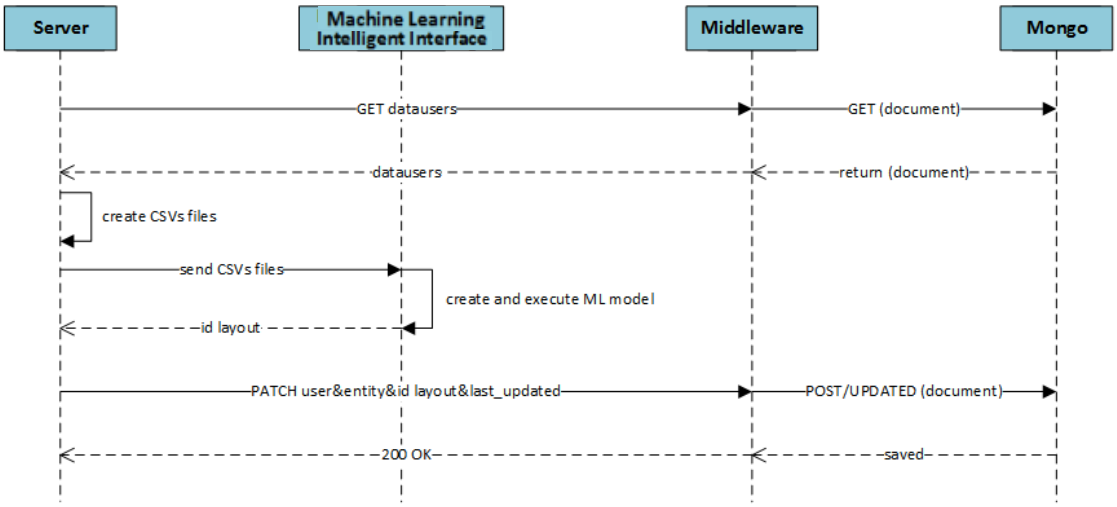

Figura 24 - Diagrama de sequência caso de uso *Machine Learning Intelligent Interface*

Este processo, representado na Figura 24, decorre *offline* e periodicamente. Cada vez que este processo é executado, o servidor efetua um pedido GET ao *middleware* para poder aceder aos dados com informações de todas as interações de todos os utilizadores. Os documentos são retornados e os dados relativos às interações são enviados para o servidor. O servidor é responsável

por colocar esses dados em ficheiros .csv, formato lido pelo modelo de ML. Seguidamente envia para o "*Machine Learning Intelligent Interface*" os ficheiros .csv gerados, um ficheiro .csv com todas as interações de todos os utilizadores e os restantes ficheiros .csv correspondem às interações de cada utilizador para cada entidade. O "*Machine Learning Intelligent Interface"* cria e executa o modelo de ML, retornando o id do *layout* atribuído a cada utilizador para cada entidade.

O servidor envia um ficheiro .json para o *middleware*, com o *username*, as entidades com os respetivos *layouts* atribuídas pelo modelo de ML e a data/hora que o modelo executou para este utilizador. A data e hora a que o modelo executou para cada utilizador é importante pois, o modelo só executa novamente para este utilizador se este realizou novas interações com a interface da aplicação depois dessa data/hora. O *middleware* efetua um pedido POST adicionando esta informação em documento à coleção do Mongo que guarda os ids dos *layouts* de cada utilizador, caso o utilizador ainda não tenha *layouts* atribuídos. Caso o utilizador já tenha *layouts* atribuídos é realizado uma atualização aos valores dos ids. Por último, é enviada uma mensagem de "ok" para o servidor.

4) *Login* de utilizador com *layout* atribuído

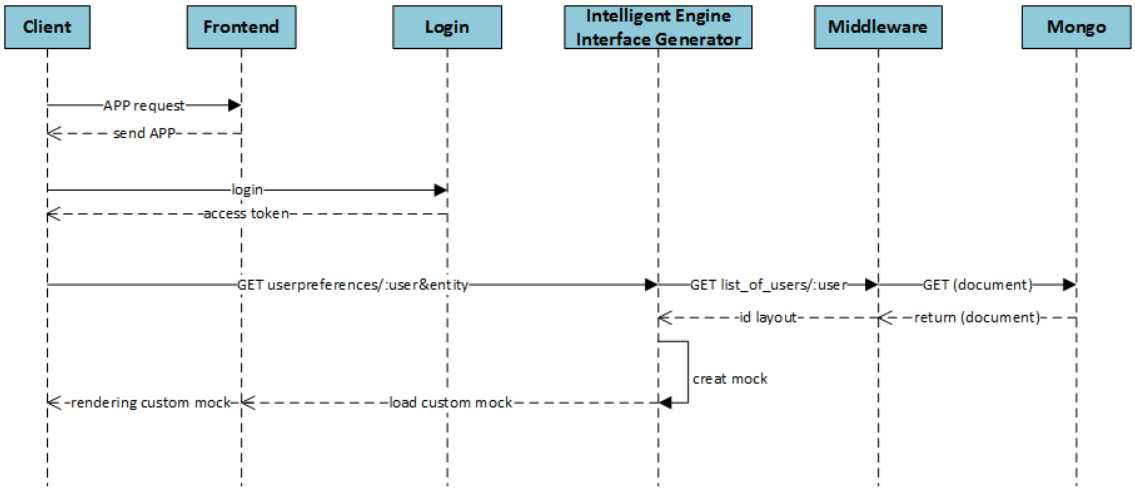

Figura 25 - Diagrama de sequência caso de uso *login* de utilizador com *layout* atribuído

Na Figura 25, é possível observar que, primeiramente é efetuado o pedido da aplicação ao servidor na aplicação *frontend* por parte do cliente. Seguidamente é devolvida a aplicação face a esse pedido e o utilizador pode efetuar o *login* na aplicação, sendo posteriormente enviado um *token* de acesso.

Quando o utilizador seleciona uma entidade disponível na aplicação, é realizado um pedido GET para o serviço "*Intelligent Engine Interface Generator*" para obter as preferências *layout* do utilizador para a entidade selecionada. Por sua vez, o serviço necessita fazer um pedido GET através de *endpoint* a um *middleware* para aceder à lista de utilizadores e respetivos ids *layouts*. Cada

utilizador pode ter no máximo um *layout* para cada entidade da aplicação. O *middleware* efetua o pedido GET do documento Mongo correspondente e é retornado o documento que contem essa informação. O *middleware* envia os ids dos *layouts* atribuídos a este utilizador para o "*Intelligent Engine Interface Generator*", que cria o *mock* e envia o *load* do *mock* personalizado para a aplicação *frontend* que efetua o *rendering* do *mock* personalizado.

As diferentes fases do modelo de *Machine Learning* decorrem maioritariamente no segundo bloco e uma pequena parte (tratamento de dados) ainda no primeiro bloco, antes de os dados serem guardados na base de dados.

4.3 Recolha e Armazenamento de Dados

Para ser possível treinar o modelo, é necessário, em primeiro lugar, conseguir recolher informação relevante e decisiva para a adaptação, acerca da interação de cada utilizador com a interface. Este conjunto de dados recolhidos a cada interação, tem de ser capaz de permitir oferecer informação relevante ao modelo de ML, de modo a que as suas previsões possam dia a pós dia oferecer a cada utilizador, individualmente, uma interface cada vez mais adaptada.

Neste sistema, optou-se por seguir uma abordagem de recolha de dados não intrusiva. Ou seja, o utilizador não é importunado com questões, que cujas respostas dadas, possam revelar o "perfil" do utilizador, como os seus gostos pessoais, etc. Foi adotada uma abordagem discreta, onde são recolhidos dados apenas relativos às interações que os utilizadores vão tendo com a interface, de modo a obter um padrão de comportamento e de preferências. Por exemplo, que elementos da interface ou que características de elementos o utilizador selecionou.

Este bloco é responsável por, para cada utilizador com *login* na aplicação, recolher os seguintes dados de *input* durante as interações: nome de utilizador, nome da aplicação que este está a utilizar e da entidade selecionada. Estes dois últimos influenciam os restantes *inputs*, pois dependendo da aplicação em que o utilizador se encontra e da entidade que selecionou é que poderá poder escolher os restantes *inputs*, como a seleção de *widget*, por exemplo certas entidades da aplicação podem não disponibilizar a escolha de *widget*. São ainda *inputs* o nível de zoom da página que o utilizador seleciona, o número de colunas de uma tabela, o tipo de ordenação das colunas da tabela, o número de registos por página da tabela e a escolha ou não da utilização de *sidebar* para efetuar filtragens.

Em seguida, é explicado em mais detalhe a relevância nas escolhas destes *inputs*.

- Utilizador

Inserir o nome de utilizador na recolha de dados é muito relevante porque é necessário saber a que utilizador corresponde os dados de *input* para efetuar posteriormente a previsão do *layout* para cada utilizador.

- Aplicação

O nome referente à aplicação é também um *input* importante à adaptação, pois para além do utilizador, é necessário saber a que aplicação é que correspondem os dados das interações para posteriormente a adaptação ser aplicada nas interfaces desta.

- Entidade

A entidade é uma representação de um conjunto de informações sobre determinado conceito da aplicação. Toda a entidade possui atributos, que são as informações que referenciam a entidade. A entidade é assim, como a aplicação, outro *input* relevante para a adaptação, pois existem vários *inputs* que dependem deste.

- Zoom

O zoom indica em valor decimal, (este valor decimal corresponde a um cálculo efetuado, de modo a que este valor seja igual em todos os computadores) o grau de zoom efetuado na janela pelo utilizador. Este *input* é relevante, pois é possível perceber se a pessoa apresenta algumas dificuldades visuais para com a interface, e então esta pode adaptar-se e por exemplo, apresentar elementos ou letras com dimensões maiores.

- *Widget* (card/tabela)

O *widget* (*card*/tabela) foi o componente da WebCT escolhido para ajustar-se a diferentes comportamentos dos utilizadores de forma dinâmica. A escolha do *widget*, indica como o utilizador prefere visualizar a informação. Esta escolha pode depender de vários fatores, como o gosto pessoal, tipo de dados ou a dimensão da quantidade de dados. Certo tipo de dados pode ser visualmente mais atrativo ser disponibilizado ao utilizador em forma de *card* do que em forma de tabela, assim como o contrário.

- Número de Colunas

O número de colunas permite interpretar a dimensão dos dados a nível de atributos. Possivelmente tabelas com muitas colunas não são muito atrativas visualmente, e por esse motivo, pode ser mais adequado disponibilizar esta informação em vista *card*.

- Número de registos por página de tabela

O número de registos de tabela disponibilizados por página, é uma escolha que a interface permite ao utilizador escolher a quantidade de dados que a tabela exibe por página. Este *input* é interessante, pois pode contribuir para uma interface mais adaptável. Sabendo a quantidade de dados que normalmente um utilizador prefere ver por registo, seria agradável no momento em que a tabela é disponibilizada ao utilizador, o número de registos por página, ser o que normalmente o utilizador prefere.

- Ordenação por coluna

Assim como número de registos por página de tabela, a ordenação da tabela por tipo de coluna, seria uma mais-valia para atribuir a cada utilizador uma interface mais adaptável e personalizada.

- *Sidebar* de filtragem

A interface disponibiliza ao utilizador um *sidebar* que permite pesquisa/filtragem de dados. Por *default*, o *sidebar* é disponibilizado maximizado (aberto/expandido). O utilizador pode ou não minimizar o *sidebar*. Através do comportamento do utilizador com este elemento, é possível oferecer a interface mais adaptável.

Na Figura 26, é possível visualizar a aplicação (portal) ARM e os elementos com os quais o utilizador pode interagir.

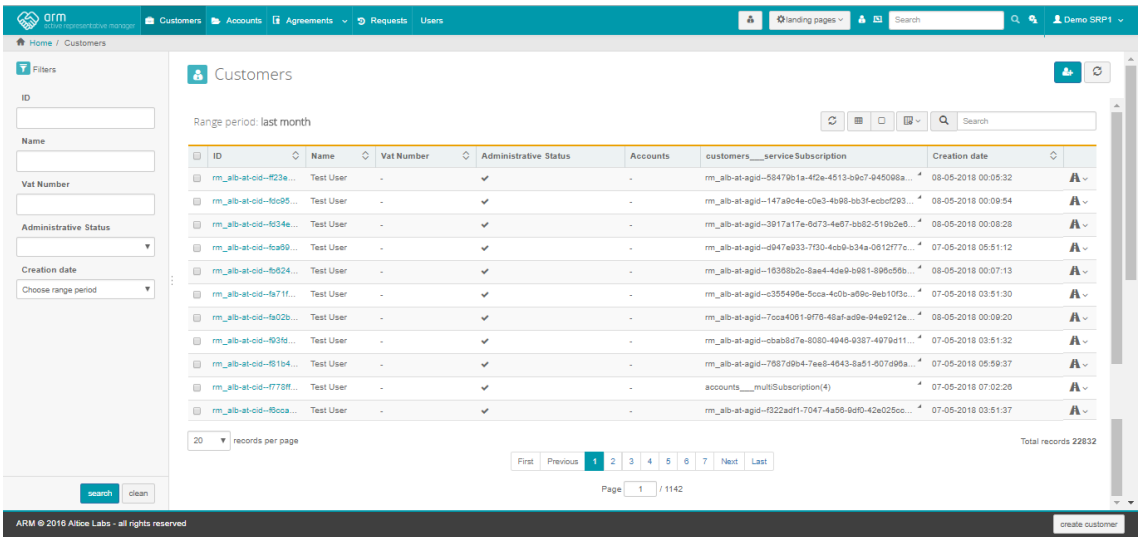

Figura 26 - Aplicação ARM

Após a recolha destes *inputs* a cada interação que um utilizador tem com a interface, é necessário armazenar os dados.

Armazenamento de Dados

Os dados são guardados numa base de dados MongoDB e o armazenamento destes é realizado através de *endpoints*.

De modo a permitir o armazenamento dos dados, foram criadas as seguintes coleções:

- **data_users:** dados das interações de todos os utilizadores;
- **list_of_users:** dados de utilizadores que já possuem dados de interações;
- **layouts:** dados relativos a todas as combinações *layouts* possíveis;
- **elements:** dados relativos aos códigos dos elementos que compõem os *layouts*;
- **execution:** dados relativos à execução.

Coleção "Data_users"

Contêm a informação relativa a todos os *inputs* de cada interação de todos os utilizadores (Tabela 5).

	Tipo	Descrição
username	Nominal	*Username* do utilizador
application	Nominal	Aplicação em utilização
zoom	Categórica	Zoom da página
entity	Nominal	Entidade selecionada na aplicação
card	Categórica	*Widget card*
tabela	Categórica	*Widget* tabela
records per page	Categórica	Registos por página da tabela
number of columns	Categórica	Número de colunas da tabela
order by	Categórica	Tabela ordenada por tipo de coluna
sidebar	Categórica	*Sidebar* aberto ou fechado

Tabela 5 - Coleção de dados das interações dos utilizadores

Coleção "List_of_users"

Contêm a informação de estado de cada utilizador com *login* efetuado, onde o *id é o username* do utilizador*, last*_update representa a data da última vez que o modelo de ML foi executado para este utilizador e o *layout* indica o *layout* atribuído pelo modelo da última vez que este foi executado (Tabela 6).

	Tipo	Descrição
_id	Nominal	*Username* do utilizador
last_update	Contínua	Data em formato milisegundos
layout	Discreta	Número do *layout* atribuído pelo modelo

Tabela 6 - Coleção de dados relativos aos utilizadores

Coleção "Layouts"

Representa todas as combinações de *layout* possíveis em formato binário, tendo em conta os *inputs* recolhidos (Tabela 7). Onde "1" indica que o *layout* contém essa característica e "0" indica que este *layout* não possui esta característica. O valor "n" representa os valores NaN. Por exemplo, quando é selecionado *card*, os *inputs* relativos à tabela têm valor "n".

	Tipo	Descrição
_id	Discreta	*Username* do utilizador
zoom	Discreta	Zoom da janela
card	Discreta	*Widget card*
tabela	Discreta	*Widget* tabela
number_of_columns_0	Discreta	Número de colunas da tabela entre [1,10]
number_of_columns_1	Discreta	Número de colunas da tabela entre [10, ∞]
number_of_columns_n	Discreta	NaN
ordered_by_column_Id	Discreta	Tabela ordenada por coluna Id
ordered_by_column_Name	Discreta	Tabela ordenada por coluna Nome
ordered_by_column_Number	Discreta	Tabela ordenada por coluna *Number*
ordered_by_column_default	Discreta	Tabela ordenada por *Default*
ordered_by_column_n	Discreta	NaN
records_per_page_100	Discreta	100 registos por página da tabela
records_per_page_20	Discreta	20 registos por página da tabela
records_per_page_50	Discreta	50 registos por página da tabela
records_per_page_n	Discreta	NaN
application_ARM	Discreta	Aplicação ARM
entity_accounts	Discreta	Entidade *Accounts*
entity_agreements	Discreta	Entidade *Agreements*
entity_customers	Discreta	Entidade *Customers*
entity_requests	Discreta	Entidade *Requests*
sidebar_false	Discreta	*Sidebar* aberto
sidebar_n	Discreta	NaN
sidebar_true	Discreta	*Sidebar* fechado

Tabela 7 - Coleção de *layouts*

Coleção "Elements"

Contêm código de todos os elementos necessários para compor uma combinação de um *layout* (Tabela 8). Desde código de componentes (card/tabela) por entidades (*customers*, *accounts*, *agreements*, *requests*), assim como o código de pequenos elementos como o número de registos por página de tabela (*records per page)*, *sidebar*, etc.

	Tipo	Descrição
records-50	Documento	Código relativo à configuração de 50 registos por página de tabela
ordered_by_column-Id	Documento	Código relativo à configuração de ordenação de tabela por Id
sidebar-close	Documento	Código relativo à configuração do *sidebar* de filtragem fechado
...	...	...

Tabela 8 - Coleção de elementos

Coleção "Execution"

Contêm dados relativos às datas/horas de execução do modelo de ML que será executado periodicamente (Tabela 9). Este valor é guardado numa coleção para que não se perca sempre que o servidor é executado.

	Tipo	Descrição
last_update	Data	Data/Hora relativa à última vez que o modelo de ML executou

Tabela 9 - Coleção de dados de execução

Tratamento de dados

Normalmente os dados necessitam, de ser submetidos a vários processos de tratamento de dados de modo a apresentarem um formato que seja interpretável por parte do algoritmo de ML escolhido.

Alguns algoritmos necessitam que os dados se encontrem na forma de valores numéricos, enquanto outros modelos só recebem valores de atributos categorizados.

A discretização de valores atribuídos é incluída no processo de tratamento/transformação de dados. É definida como um processo de conversão de valores de atributos de dados contínuos num conjunto finito de intervalos, que posteriormente, é associado a cada intervalo uma categoria [100]. Normalmente,

os dados categóricos necessitam ser codificados para binário, porque a maior parte dos algoritmos de ML obtém melhores resultados a nível de performance quando os números variam entre 0 e 1 [101].

Devido ao modelo de ML escolhido a utilizar neste sistema, é então aplicado o processo de discretização aos dados contínuos de modo a facilitar a interpretação dos dados pelo algoritmo e a posterior transformação dos mesmos para valores binários. Os *inputs* de valores contínuos são agrupados em intervalos de valores e a cada intervalo é atribuído uma classe/rótulo (*label*). Os rótulos/classes dos intervalos substituem os valores.

O zoom apresenta um valor decimal e considerando que o valor 0,81 corresponde a um nível de zoom de página relevante, os valores foram divididos em duas categorias:

Categoria	Zoom
0	< = 0.8
1	> 0.8

Tabela 10 - Categorias do atributo zoom

O número de colunas, valores numéricos inteiros maiores que zero são também distribuídos em dois grupos:

Categoria	Nº Colunas
0	< 10
1	> = 10

Tabela 11- Categorias do atributo número de colunas

Este processo é realizado antes de cada interação guardada. Após o processo de discretização e antes de serem guardados na base de dados, os dados, podem apresentar as seguintes possibilidades de categorias:

- Zoom

Categoria	Zoom
0	[0 , 0.8 [
1	[0.8 , ∞ [

Tabela 12 - Categorias do atributo zoom

- *Card*

Categoria	Widget
True	*card*
False	tabela

Tabela 13 - Categorias do atributo *widget card*

- Tabela

Categoria	Widget
True	tabela
False	*card*

Tabela 14 - Categorias do atributo *widget table*

- Número de colunas

Se o *widget* selecionado for *card*, pois este atributo de *input* é relativo apenas ao *widget* tabela, é atribuída a categoria "n". N representa os números NaN (*missing values*), visto que os algoritmos de ML interpretam estes valores de diferentes maneiras, sendo que alguns removem as linhas de dados que contenham dados do tipo NaN. Por este motivo, e porque neste caso a ausência deste atributo significa que o *widget* selecionado é *card* e esta informação não pode ser removida, os valores NaN devem ser tratados antes de serem enviados para o modelo.

Categoria	Nº Colunas
0	[0, 10 [
1	[10, ∞ [
n	Se *widget card*

Tabela 15 - Categorias do atributo número de colunas

- Número de registos por página de tabela

Categoria	Nº de registos por página da tabela
20	20
50	50
100	100
n	Se *widget card*

Tabela 16 - Categorias do atributo número de registos por página de tabela

- Ordenação por coluna

Categoria	Ordenação da tabela por
Id	Id
Name	Name
Number	Number
Default	Se não é escolhida nenhuma ordenação
n	Se *widget card*

Tabela 17 - Categorias do atributo ordenação de colunas

- *Sidebar* de filtragem

Categoria	*Sidebar* de filtragem
True	fechado
False	aberto
n	Se *widget card*

Tabela 18 - Categorias do atributo *sidebar* de filtragem

Após este processo, os dados são guardados na base de dados e antes de serem enviados para o modelo de ML devem estar completamente adaptados a um formato interpretável pelo algoritmo escolhido.

4.4 *Machine Learning Intelligent Interface*

Este bloco, *Machine Learning Intelligent Interface*, é responsável por criar o modelo de ML, treiná-lo, testá-lo e fazer as previsões para novos *inputs* de dados. Primeiramente, é necessário verificar se os dados estão aptos para serem interpretados pelo algoritmo escolhido. Por isso, apesar de no bloco anterior já ter sido aplicado algum tratamento aos dados (discretização), e visto que o algoritmo escolhido lê dados binários, ainda é necessário transformar os dados para dados binários. É utilizada a biblioteca “pandas” disponibilizada pelo Python para efetuar algumas substituições nos *inputs* dos atributos *card* e tabela, substituindo as categorias de “Verdadeiro” ou “Falso” pelos números binários “0” e “1” respetivamente.

De modo a finalizar o processo de adaptação e tratamento de dados é utilizada a função “get_dummies” providenciada pela biblioteca “pandas” para transformar os dados categóricos em dados numéricos equivalentes. Onde “1” representa a presença de uma categoria e “0” representa a sua ausência.

Por exemplo (Tabela 19), quando a categoria *sidebar* está configurada a *true*, o modelo de ML representa [1,0,0]:

Categoria	***Sidebar* de filtragem**
1	Fechado (*true*)
0	Aberto (*false*)
0	Se *widget* card (n)

Tabela 19 - Categorias do atributo *sidebar* de filtragem

Apesar de os dados relativos aos atributos iniciais serem numéricos ou categóricos, os modelos de ML obtêm as representações desses dados em valores numéricos ou vetores numéricos.

O modelo recebe o conjunto de dados, relativos às interações de todos os utilizadores com a interface, já tratados. Seguidamente, os dados são separados em dois conjuntos, um conjunto de dados de treino com 70% dos dados e outro conjunto de dados de teste com 30% dos dados.

O conjunto de treino é um conjunto de dados utilizado para construir o classificador (modelo), e quanto maior for este conjunto, o classificador obterá melhores resultados, aumentando a probabilidade de assertividade na fase de teste. O conjunto de teste é utilizado para estimar a qualidade do classificador, avaliando a precisão e a eficácia do modelo.

Por fim, através da análise preditiva é possível obter a classe mais adequada para um utilizador em específico. A classe que queremos "prever" corresponde a uma combinação *layout*.

A criação de um modelo de ML resulta nas seguintes etapas:

1) Recolha de dados;
2) Tratamento de dados;
3) Escolha do algoritmo;
4) Fase de treino;
5) Fase de teste;
6) Previsão.

As etapas de tratamento dos dados e da escolha do algoritmo estão implicitamente ligadas, visto que a escolha do algoritmo é que influencia o tratamento de dados adequado.

Tendo em conta que a primeira etapa e a segunda já foram explicadas anteriormente, seguidamente serão explicadas, em mais detalhe, as restantes etapas.

4.4.1 Escolha do Algoritmo

A escolha do algoritmo para o modelo depende de vários fatores, tais como o tipo de dados, dimensão dos dados, etc.

Como referido anteriormente, a Tensorflow providencia um conjunto de modelos pré-construídos para treinar e prever dados. Dentro desse conjunto, o *Deep Neural Network Classifier* foi o escolhido, porque é ideal para modelos "*deep"* que realizam classificações multi-classe através de redes neurais avançadas. Visto que, um dos principais problemas/objetivos, desta dissertação, consiste em fornecer a interface mais adequada para cada utilizador e que os *layouts* são a interface de um sistema, o que se pretende é realizar classificações de *layouts*. Tendo em conta que as interfaces serão personalizadas a cada utilizador, o número de combinações *layout* é vasto, pois depende das combinações possíveis dos dados recolhidos nas interações que os utilizadores têm com os componentes. Por este motivo, encaixa-se num problema de classificação multi-classe. Uma classificação multi-classe, consiste em classificar uma dada entrada de dados em várias classes/rótulos. Neste problema, cada classe representa um *layout*.

4.4.2 Fase de Treino

Primeiramente, são calculados o número de linhas e colunas de cada conjunto (informação necessária para o modelo) e são gerados os ficheiros .csv de treino e teste. Seguidamente, são carregados os dados contidos nos ficheiros .csv de treino e teste para o módulo tf.contrib.learn.datasets da Tensorflow. É necessário indicar o nome do ficheiro a carregar, o tipo do "*target*", ou seja, o tipo da classe a prever, que neste caso é um valor inteiro (np.int32) que representa o id do *layout*, o tipo de dados das "*features*" (colunas), e a posição (coluna) no ficheiro .csv onde se encontra a classe a prever.

```
training_set=tf.contrib.learn.datasets.base.load_csv_with_header (
      filename= 'train.csv',
      target_dtype = np.int32,
      features_dtype = np.int32,
      target_column = 0)
```

Posteriormente, é instanciada a Tensorflow (`tf`) e é indicado a esta que se pretende construir um estimador do tipo *Deep Neural Network Classifier* com três

camadas, com dez, vinte e dez unidades respetivamente, indicando assim à Tensorflow que se pretende construir uma rede neural com três camadas, onde cada camada contem respetivamente dez, vinte e dez neurônios (nós). O número de camadas e respetivos nós, foi escolhido com base nos resultados dos testes obtidos com atribuição de vários valores a cada parâmetro configurável. Para proceder à criação do classificador é necessário indicar ainda as *features* a utilizar no modelo, assim como as *hidden units*, que representam a lista de unidades escondidas por camada sendo que todas as camadas se encontram completamente conectadas.

Por fim, é ainda necessário indicar o número de classes possíveis de prever, o que corresponde ao número de combinações possíveis de *layouts*, e o model_dir que é um diretório que permite guardar os parâmetros do modelo, grafo, etc., e pode ser utilizado para carregar *checkpoints* do diretório num estimador para continuar a treinar um modelo anteriormente guardado, sem ter de ser necessário começar a treinar o modelo novamente do início.

```
classifier = tf.estimator.DNNClassifier(
feature_columns = feature_columns,
hidden_units = [10,20,10],
n_classes = 771,
model_dir = “/tmp/intelligent_mode”)
```

Os *inputs* do conjunto de treino e do conjunto de teste, são definidos em variáveis x e y criando, através do `tf.constant` um tensor constante, que irá ser populado pelos dados dos conjuntos.

Através do método `train` disponibilizado na Tensorflow é possível treinar o modelo. Este método recebe os *inputs* dos dados do conjunto de treino (Figura 27) e é indicado o número de *steps*, que indica o número de vezes que se pretende que o estimador realize o ciclo de treino, ou seja o número de vezes que percorrerá o conjunto de dados de treino. O número de *steps* atribuído foi dois mil, devido aos resultados obtidos dos testes feitos aos parâmetros configuráveis do modelo.

```
classifier.train(input_fn = get_train_inputs, steps=2000)
```

```
481,0,1,0,1,0,1,0,0,0,1,0,0,1,0,0,0,0,1,0,1,0,0,0
63,0,0,0,1,0,0,1,0,1,0,0,0,1,0,0,0,0,1,0,0,1,0,0
647,0,1,0,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,1,0,0
639,1,0,0,1,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0
279,0,1,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,1,0,0
2,1,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1
```

Figura 27 - Exemplo de ficheiro do conjunto de dados de treino

4.4.3 Fase de Teste

Após o treino, é necessário testar o modelo, por isso, é calculada a *accuracy*, que indica o grau de assertividade do modelo. A função de calculo da *accuracy*, utilizada pela tensorFlow, cria duas variáveis locais ("total" e "count"), que são utilizadas para calcular a frequência com que as previsões acertam na classe. Essa frequência é retornada como *accuracy*, uma operação que divide o número "total" de previsões de classes acertadas pela "count", que representa o número de previsões efetuadas [102]. De modo a calcular a *accuracy*, utiliza-se `classifier.evaluate` que recebe os *inputs* do conjunto de teste (Figura 28) assim como a indicação do número de *steps* que se pretende utilizar para a realização do teste, que neste caso é apenas um *step* pois só se pretende testar uma vez. Os dados do conjunto de teste são então comparados com os dados do conjunto de treino e é atribuído um *layout* a cada interação nos dados do conjunto de teste. Esse *layout* atribuído, é comparado com o valor *layout* verdadeiro. O valor da *accuracy*, varia entre zero e um, e depende do número de classes do conjunto de dados de teste que o modelo acertou. Se o modelo acertar todos os *layouts* para todos os *inputs* de dados do conjunto de teste então a *accuracy* vai ter valor de 1.

```
accuracy = classifier.evaluate(input_fn = get_test_inputs, steps=1)
```

Se o conjunto de dados de teste contiver *inputs* de dados que não se encontram no conjunto de dados de treino, então o modelo não vai acertar na classe a prever, pois não tem conhecimento suficiente, no entanto retornará um *layout* cujo peso atribuído pelo algoritmo é muito idêntico. A combinação de dados que representa um *layout* é uma combinação de dados binários (Figura 28).

```
?,0,0,0,1,0,0,1,0,1,0,0,0,1,0,0,0,0,1,0,0,1,0,0
?,1,1,0,1,1,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,1
?,1,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1
?,1,1,0,1,0,0,0,1,0,0,1,0,1,0,1,0,0,0,0,0,0,0,1
```

Figura 28 - Exemplo de ficheiro do conjunto de dados de teste

4.4.4 Previsão

Após treinar e testar o modelo, é possível realizar a previsão da classe *layout*, baseado em dados sem classes atribuídas. A previsão de novos dados desconhecidos pelo modelo e sem classes atribuídas pode ser aplicada através `classifier.predict` que recebe como argumento, um *array* de *arrays* com o conjunto de *inputs* de um determinado utilizador sem *layouts* atribuídos e devolve um único *array* com os valores dos ids dos *layouts* atribuídos para cada

combinação de dados de *input*, ou seja para cada interação que o utilizador teve com a interface é previsto o *layout* .

```
[[1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0], [1, 0, 1, 1, 0, 0, 1, 0,
 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1], [1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0], [1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
 0, 1, 0, 0, 0, 1]]
       New Samples, Class Predictions:    [2, 351, 2, 415, 2]
```

Figura 29 - Exemplo de previsão de *layouts*

Como é possível verificar neste exemplo, representado na Figura 29, a predição recebe um *array* com quatro *arrays*, e cada um corresponde a uma interação que o utilizador teve com a interface. Para cada interação o modelo calcula o id do *layout* adequado. Neste exemplo, os ids retornados são o *layout* com id=2, o id=351, o id=2, o id=415 e o id=2.

Tendo acesso a esta informação, é possível identificar através do cálculo da moda, o *layout* mais comum e predominante para este utilizador. No caso de não existir moda, é disponibilizado, o *layout* que está atribuído ao utilizador naquele momento. A Figura 30, indica que o *layout* predominante neste exemplo, é o *layout* cujo o id é o 2.

```
layout = mode(predictions)
```

```
arm-admin----->LAYOUT FROM ALGORITHM: 2
```

Figura 30 - *Output* previsão de *layout*

Estes novos dados com classe calculada pelo modelo são adicionados ao ficheiro .csv que contêm os *inputs* de todos os utilizadores e o id correspondente do *layout* mais comum para cada entidade é guardado na coleção “list_of_users” da base de dados (Tabela 20), assim como a data/hora da última vez que o modelo foi executado (last_updated) para os dados deste utilizador.

user	***layout***	**last update**
Jorge	2	1525110224224
Srp1	20	1525110224224
...	...	...

Tabela 20 - Exemplo simplificado da coleção da lista de informações de utilizadores

4.5 *Intelligent Engine Interface Generator*

Este bloco executa quando um utilizador efetua *login* na aplicação e é responsável por construir e enviar, em tempo real, o *mock* para a WebCT para que esta possa efetuar o *load* e disponibilizar a interface ao utilizador.

Primeiramente, quando o utilizador efetua *login* na aplicação e seleciona uma entidade, através de um *endpoint* é verificado na base de dados se já existe um *layout* atribuído. Se não existir *layout* atribuído para este utilizador e para a respetiva entidade selecionada por este, então é retornado para a WebCT um ficheiro .json indicando que não tem *layout* atribuído, e esta, interpreta-o e faz *load* do *mock default* para esta entidade e disponibiliza-o ao utilizador. O *mock default* corresponde à combinação *layout default* que a WebCT disponibiliza para todos os utilizadores.

No caso de o modelo de ML já ter atribuído um *layout* para este utilizador e para a respetiva entidade selecionada por este, o *Intelligent Engine Interface Generator* acede à coleção que contêm a combinação binária desse *layout* (Tabela 21) e interpreta-a de modo a identificar os elementos presentes nesta. Após a identificação dos elementos, acede a outra coleção (Tabela 22) que contêm os elementos e as respetivas configurações de código correspondentes.

***layout* id**	**combinação**
63	[1,1,0,0,1,1,0,1,0,1,0,1,0,0,0,0,0,1,0,0,1,1]
1	[1,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,1,0,0,1,0]
...	...

Tabela 21 - Exemplo simplificado da coleção de informações sobre ids de *layouts*

element	**cfg**
card	{...}
table	{...}
zoom 1	{...}
zoom 0	{...}
...	...

Tabela 22 - Exemplo simplificado da coleção de elementos

O *Intelligent Engine Interface Generator* acede a um *mock* geral (Figura 31), formato base a todas as entidades que só contêm as chaves comuns entre todas e verifica-se as configurações dos elementos que tem as mesmas chaves, se contiver, as chaves e os respetivos valores, é construído um novo ficheiro *mock* personalizado para este utilizador. Se as configurações apresentarem chaves que não existem no *mock* geral, estes são adicionadas posteriormente ao *mock* personalizado.

```
{
        “type” : ”page”,
        “id” : ”pageConfiguration”,
        “navigation” : “”,
        “header” : “”,
        “tabs” : “”
```


Figura 31 - Mock geral

A Figura 32, representa de forma esquematizada o processo referido em cima. No canto esquerdo, na parte superior está a representação do *mock* geral, do lado direito está uma representação do *mock default* da entidade *agreements* e do lado esquerdo em baixo uma representação do *sidebar* de filtragem configurado fechado.

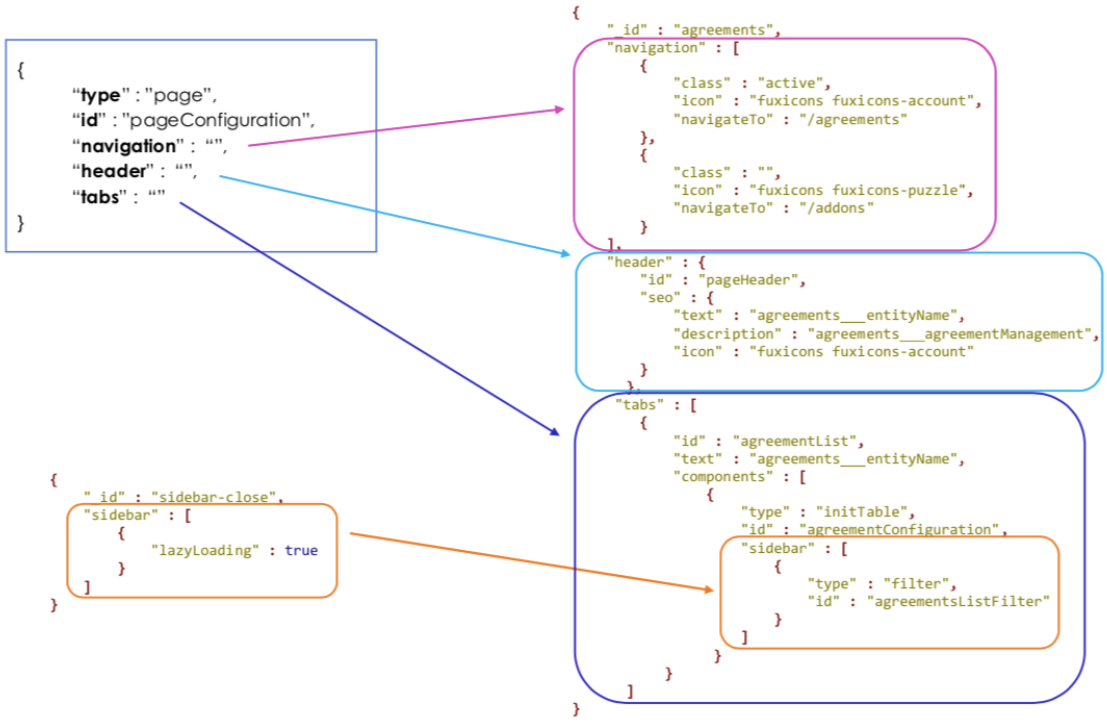

Figura 32 - Processo Intelligent Engine Interface Generator

Neste exemplo simples, primeiramente, acede-se ao *mock default* da entidade *agreements*, disponibilizado na coleção de elementos da base de dados e verifica-se as chaves principais com as chaves principais do ficheiro *mock* geral. É gerado um *mock* personalizado com as chaves do *mock* geral. As chaves do ficheiro *default* que coincidirem, são adicionadas ao *mock* personalizado o valor correspondente. Seguidamente, é percorrido o *mock* personalizado, de modo a verificar todas as subchaves. O elemento *sidebar* está configurado a “fechado”, e o *mock default* apresenta sempre o *sidebar* filtragem aberto, por isso é necessário adicionar as subchaves da representação do

sidebar de filtragem configurado a fechado dentro da subchave "*sidebar*" do *mock* personalizado.

Por fim, é obtido o ficheiro .json do *mock* personalizado (Figura 33).


```
{
"type" : "page",
"id" : "pageConfiguration",
"tabs" :  [
      {
         "id" : "agreementList",
         "text" : "agreements___entityName",
         "components" : [{
               "type" : "initTable",
               "id" : "agreementConfiguration",
               "sidebar" : [
                  {
                     "type" : "filter",
                     "id" : "agreementsListFilter",
                      "lazyloading": "true"

                  }
               ]
         }]
      }
],
"navigation" :  [
      {
         "class" : "active",
         "icon" : "fuxicons fuxicons-account",
         "navigateTo" : "/agreements"
      },
      {
         "class" : "",
         "icon" : "fuxicons fuxicons-puzzle",
         "navigateTo" : "/addons"
      }
   ],
"header" :  {
      "id" : "pageHeader",
      "seo" : {
         "text" : "agreements___entityName",
         "description" : "agreements___agreementManagement",
         "icon" : "fuxicons fuxicons-account"
      }
 }
}
```


Figura 33 - Exemplo simples de um mock personalizado

A última etapa deste processo, consiste em retornar para a *framework* WebCT, este *mock* personalizado e gerado dinamicamente em tempo real, para que este possa efetuar o *load* e disponibilizá-lo, oferecendo assim uma interface adaptável e personalizada a cada utilizador. Este sistema de geração dinâmica de *mocks* permite que seja apenas necessário a existência de um *mock* geral, em vez de ser necessário existir um *mock* para cada combinação *layout*.

5 Resultados

Este capítulo, tem como objetivo comprovar que a estrutura implementada permite responder aos desafios propostos. Assim como permite estudar os parâmetros do modelo, que oferecem melhor desempenho e performance.

Como referido anteriormente, não existem valores específicos que se podem atribuir aos parâmetros dos modelos de ML. Para cada problema, é necessário ir testando o modelo com vários valores em cada parâmetro, de modo a ser possível obter resultados cada vez mais otimizados.

Posto isto, e de modo a permitir o mesmo ambiente para todos os testes, para cada teste, foi utilizado o mesmo conjunto de dados e também o mesmo conjunto de novas entradas a utilizar na previsão. A fase de previsão recebe sempre os dados do mesmo utilizador e são realizadas três previsões, ou seja, conjuntos de novos dados de interações do mesmo utilizador, mas para três entidades da aplicação (*customers*, *accounts* e *aggreements*). É aplicada uma previsão para cada uma das três entidades de modo a obter o *layout* mais adequado para este utilizador em cada uma das entidades.

Nas tabelas seguintes, a última coluna indica quantos *layouts* o modelo acertou para um novo utilizador. Este valor não se reflete no valor da *accuracy,* porque o valor da *accuracy* indica quantos *layouts* do conjunto de dados de teste (com dados das interações de todos os utilizadores) o modelo acertou.

Nos testes, foi aplicada apenas variância de valores ao número de camadas e de *steps* da fase de treino, visto que o modelo utilizado foi um modelo pré-construído disponibilizado pela TensorFlow.

O valor correspondente ao número de nós por camada também pode ser variado, no entanto optou-se pelos valores *standard* mencionados na documentação da TensorFlow.

- Modelo de ML com uma camada

Primeiramente foi testado o modelo com uma camada escondida de dez neurónios, com alteração do número de *steps* em 1, 100, 1000 e 2000.

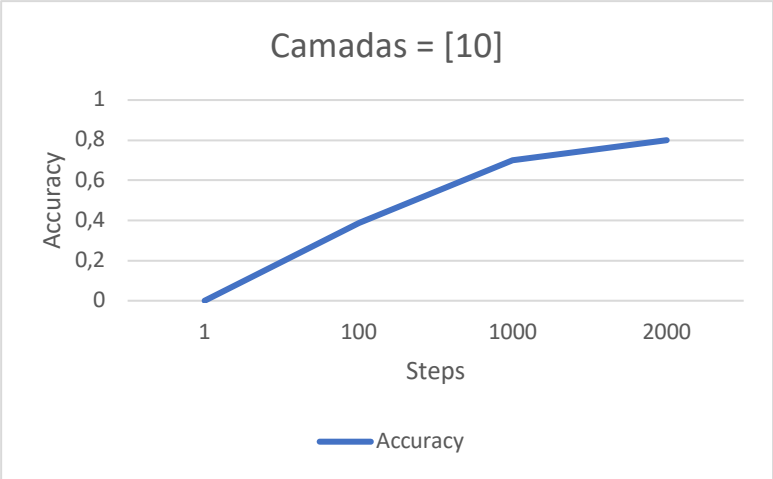

Figura 34 - Gráfico de modelo de ML com uma camada

Através do gráfico representado na Figura 34, é possível verificar que o valor da *accuracy* do modelo vai aumentando significativamente com o aumento do número de *steps*.

Camadas	*Steps*	*Accuracy*	*Tempo (minutos)*	*Layouts (corretos)*
[10]	1	0,000000	<1	0
[10]	100	0,386503	<1	1
[10]	1000	0,771686	1	2
[10]	2000	0,806056	3	2

Tabela 23 - Tabela de resultados de modelo de ML com uma camada

A Tabela 23, permite observar os resultados obtidos para estes valores dos parâmetros. É possível verificar que com uma camada e com um *step* apenas, o resultado da *accuracy* é zero, indicando que o modelo não acertou nenhuma classe do conjunto de dados de teste. Assim como, após a fase de treino e de teste, não acertou a classe de nenhum conjunto de novos dados utilizados na previsão (*layouts*). Ou seja, o modelo não acertou o *layout* mais adequando para o utilizador em nenhuma das três entidades.

Quando o número de *steps* aumenta para 100 é possível verificar que o valor da *accuracy* aumentou, mas 0,386503 não é um valor satisfatório. Para além disso o modelo só acertou um *layout*.

Quando o número de *steps* aumenta para 1000 e 2000 é possível verificar que o valor da *accuracy* aumentou substancialmente, comparativamente com os valores obtidos com 1 e 100 *steps*. O modelo conseguiu acertar duas das três previsões.

- Modelo de ML com duas camadas

Seguidamente foi testado o modelo com duas camadas de dez e vinte neurónios respetivamente, com alteração do número de *steps* em 1, 100, 1000 e 2000.

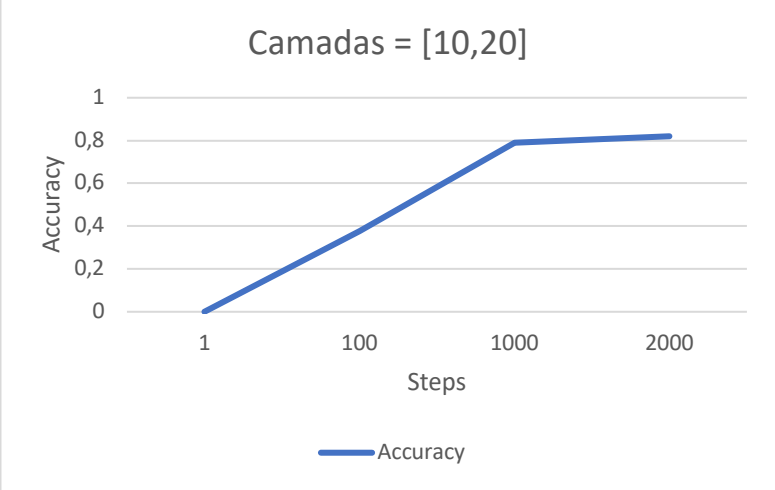

Figura 35 - Gráfico de modelo de ML com duas camadas

Através do gráfico anterior, é possível verificar que o número de *steps* na fase de treino é relevante para o desempenho do modelo.

Camadas	*Steps*	*Accuracy*	*Tempo (minutos)*	*Layouts (corretos)*
[10,20]	1	0,000818	<1	0
[10,20]	100	0,376226	<1	1
[10,20]	1000	0,797054	1	2
[10,20]	2000	0,821604	3	3

Tabela 24 - Tabela de resultados de modelo de ML com duas camadas

A Tabela 24, permite observar os resultados obtidos para estes valores de parâmetros. É possível verificar que com duas camadas e com um *step* apenas, o modelo não acertou nenhum *layout*, reforçando a importância do número de *steps*. Na fase de previsão, o modelo, apesar de já ter duas camadas, não acertou o *layout* mais adequando para o utilizador em nenhuma das três entidades.

Quando o número de *steps* aumenta para 100 é possível verificar que o valor da *accuracy* aumentou, mas 0,376226 não é um valor satisfatório e para além disso o modelo só acertou um *layout*.

No entanto, com o aumento do número de *steps* para 1000 e 2000 é possível verificar que o valor da *accuracy* aumentou para 0,797054 e 0,821604 respetivamente. Comparativamente com os valores obtidos no teste anterior,

com uma camada, também aumentou ligeiramente o valor da *accuracy*. Também é possível verificar que o modelo já conseguiu acertar as três previsões quando utilizado 2000 como valor de *steps*.

Apesar de o modelo ter acertado nos três *layouts*, o valor da *accuracy* obtido não é o valor máximo, no entanto 0,821604 já é considerado um valor aceitável. Como referido anteriormente, o valor da *accuracy* varia entre zero e um.

- Modelo de ML com três camadas

O terceiro teste, consiste em testar o modelo com três camadas de dez, vinte e dez neurónios respetivamente, com alteração do número de *steps* em 1, 100, 1000 e 2000.

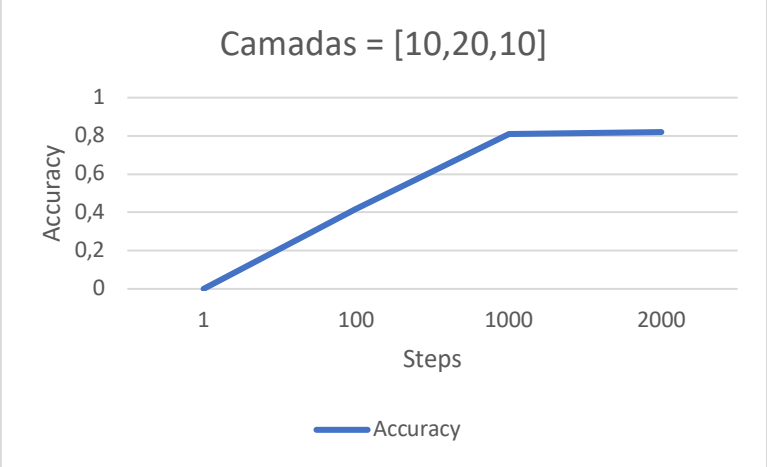

Figura 36 - Gráfico de modelo de ML com três camadas

Assim como os gráficos anteriores, é possível verificar que o número de *steps* na fase de treino é relevante para o desempenho do modelo (Figura 36).

Camadas	*Steps*	*Accuracy*	*Tempo (minutos)*	*Layouts (corretos)*
[10,20,10]	1	0,019640	<1	0
[10,20,10]	100	0,419632	<1	1
[10,20,10]	1000	0,819149	1	2
[10,20,10]	2000	0,825599	4	3

Tabela 25 - Tabela de resultados de modelo de ML com três camadas

A Tabela 25, disponibiliza os valores obtidos no teste ao modelo com três camadas. Os valores da *accuracy* aumentam ligeiramente, relativo aos dois testes anteriores com uma e duas camadas. O modelo obtém o melhor resultado de *accuracy* e de previsão quando o número de *steps* na fase de treino é 2000.

- Modelo de ML com quatro camadas

Por fim é testado o modelo com quatro camadas de dez, vinte, vinte e dez neurónios respetivamente, com alteração do número de *steps* em 1, 100, 1000 e 2000.

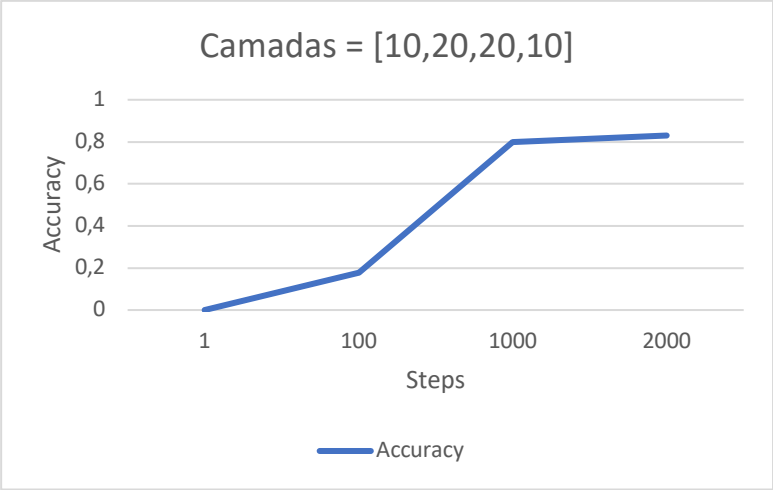

Figura 37 - Gráfico de modelo de ML com quatro camadas

Assim como os gráficos anteriores, é possível verificar que o valor da *accuracy* aumenta consoante o aumento do número de *steps*.

Camadas	*Steps*	*Accuracy*	*Tempo (minutos)*	*Layouts (corretos)*
[10,20,20,10]	1	0,000000	<1	0
[10,20,20,10]	100	0,176471	1	0
[10,20,20,10]	1000	0,809329	1	2
[10,20,20,10]	2000	0,830606	5	3

Tabela 26 - Tabela de resultados de modelo de ML com quatro camadas

Os resultados disponibilizados na Tabela 26, permitem verificar que os valores da *accuracy* não são significativamente melhores que os valores obtidos no teste anterior cujo modelo tinha três camadas. Por isso, e visto que o tempo de execução também aumenta comparativamente com o teste anterior, os resultados não justificam a utilização de quatro camadas para este problema, apesar de quando utilizados 2000 *steps* o valor da *accuracy* ser ligeiramente superior ao valor 0,825599 do teste anterior com três camadas.

Tendo em conta os resultados obtidos nos testes realizados, o modelo obteve valores mais otimizados quando o número de camadas era três e o número de *steps* 2000. Por isso, para este problema foram utilizados estes valores de parâmetros no modelo de ML aplicado a este sistema.

Após obter-se os valores dos parâmetros do modelo otimizado, procedeu-se ao estudo em cinco pessoas da empresa, que utilizam este portal, que interagissem com a aplicação durante um período de tempo. O objetivo final consistia em avaliar se para cada um dos cinco utilizadores, o modelo de ML previa as interfaces mais adequadas.

Primeiramente é apresentada a interface *default* (Figura 38). Esta interface é disponibilizada para todos os utilizadores a quem o modelo de ML ainda não atribuiu *layout*.

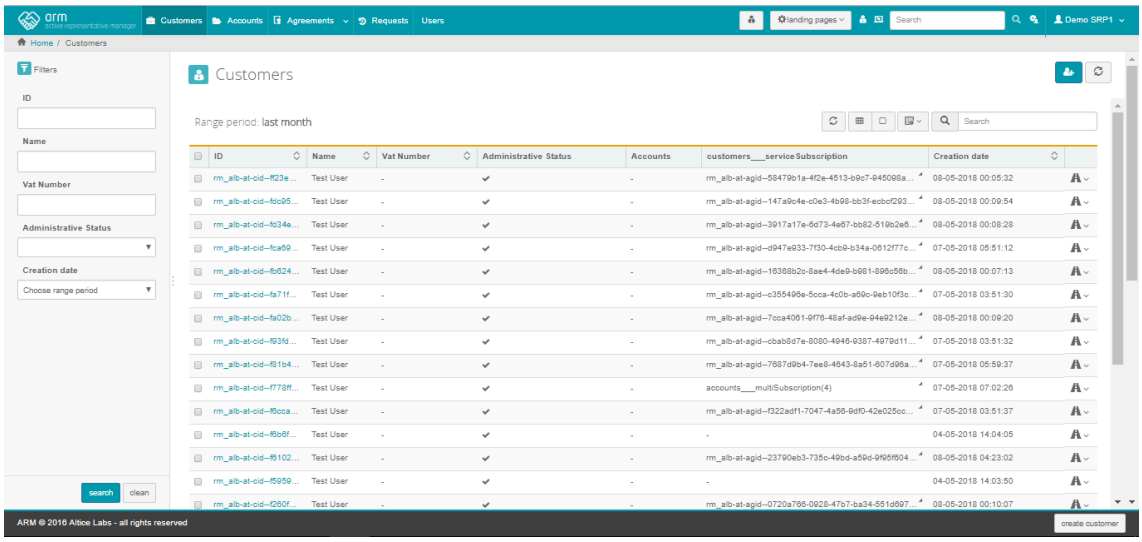

Figura 38 - Interface *default*

Seguidamente, é analisado em mais detalhe, os casos de estudo de duas (utilizador Jorge e utilizador Srp1) das cinco pessoas que interagiram com o portal.

Ao longo do período de interação com o portal, o utilizador Jorge foi selecionando interações de acordo com as suas preferências. Relativamente ao *widget* selecionado, este utilizador demonstra ter preferência por visualizar a informação em vista tabela, normalmente com ordenação por id ou ordenação por nome e com cinquenta ou cem registos por página da tabela. Em relação ao *sidebar* de filtragem, esporadicamente recorreu à filtragem.

A Figura 39, expõem a interface *layout* atribuída pelo modelo de ML ao utilizador Jorge. Esta configuração *layout*, apresenta a informação em vista tabela, ordenação da mesma por id de clientes e o *sidebar* de filtragem é disponibilizado fechado.

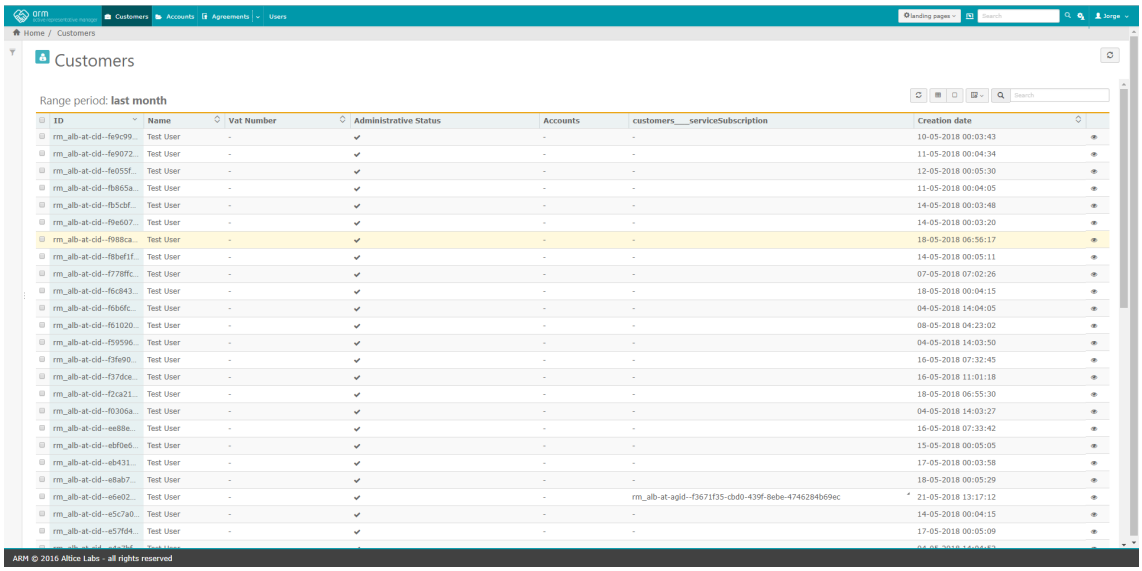

Figura 39 - Interface personalizada do utilizador Jorge

Ao longo do período de interação com o portal, o utilizador Srp1, elegeu maioritariamente visualizar a informação em vista de *card*.

A Figura 40, representa a interface *layout* atribuída pelo modelo de ML a este utilizador. Permite verificar que este, tem preferência por visualizar informação em vista *card*.

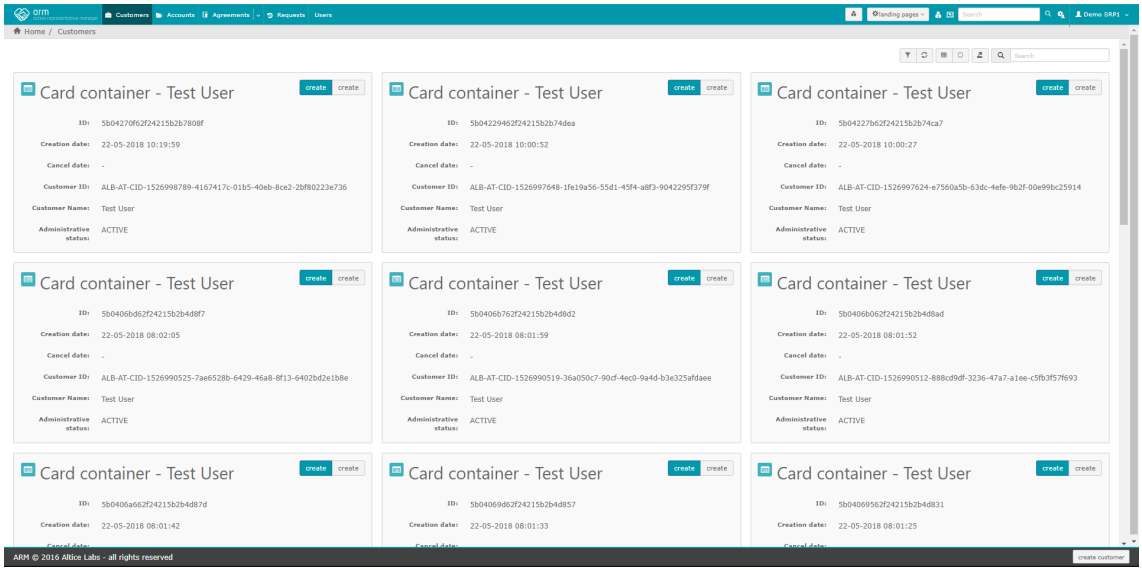

Figura 40 - Interface personalizada do utilizador Demo Srp1

O estudo permitiu então concluir que a interface atribuída a cada utilizador vai de encontro ao padrão de comportamento que cada utilizador foi tendo com o portal. A mesma aplicação/portal, consegue disponibilizar a mesma informação, mas em diferentes perspetivas de interfaces, permitindo assim oferecer acesso rápido e facilitado a serviços especializados relacionados com o perfil e preferências de cada utilizador.

6 Conclusões

Este capítulo, tem como objetivo mencionar algumas conclusões sobre o trabalho desenvolvido, problemas encontrados, trabalho a desenvolver futuramente e por fim conclusões gerais.

6.1 Conclusões

O objetivo deste estudo consistia em definir e implementar as melhores metodologias para desenvolver um sistema de geração de interfaces gráficas dinâmicas em tempo real e adaptáveis de acordo com os comportamentos do utilizador, permitindo uma melhor experiência de utilização.

Como resultado, foram desenvolvidos dois subsistemas. O primeiro, e por forma à necessidade de adaptação dos utilizadores, de forma contextualizada e inteligente, recorreu-se a tecnologia *Machine Learning*, usando algoritmos de *Deep Learning*. Este subsistema, tem como *inputs* os dados recolhidos aquando as interações entre os utilizadores e a aplicação e através de um modelo de ML, desenvolvido usando a *framework* TensorFlow, que ao processar esses dados consegue detetar padrões de comportamento dos utilizadores, sendo capaz de propor qual a configuração da interface *layout* da aplicação mais adequada. Com a informação que o subsistema referido anteriormente oferece, foi possível atribuir *layouts* personalizados a cada utilizador.

O segundo subsistema desenvolvido, ficou responsável por construir cada *layout*, identificando de forma dinâmica o resultado que será apresentado. Desta forma, o *layout* é gerado em tempo real de acordo com as preferências de cada utilizador, evitando assim a necessidade da existência de um portefólio que reúna todas as combinações possíveis de *layouts*.

Foi possível verificar, que o sistema desenvolvido, consegue de forma inovadora, criar uma personalização nas interfaces, proporcionando aos utilizadores interações mais ricas e contextualizadas e desta forma uma melhor experiência de utilização. Para além disso, consegue substituir a forma tradicional de gerar interfaces, por uma forma dinâmica e automatizada, reduzindo assim os custos.

6.2 Problemas Encontrados

Durante a realização deste estudo, foram encontrados diferentes tipos de problemas.

Perante a inexistência de estudos ou informação científica sobre como abordar a problemática das aplicações adaptáveis, a identificação do id *layout* correto, constitui um desafio.

Por outro lado, conseguir determinar os momentos indicados para recolher os dados das interações, e a sua periodicidade, de forma a melhorar o desempenho do modelo de ML, constitui um problema, que apesar de ter sido aplicada alguma calibração, melhorias poderão ser realizadas futuramente.

Por fim, na implementação do subsistema "*Intelligent Engine Interface Generator"* a escolha sobre a melhor abordagem a adotar para gerar as interfaces dinâmicas, tendo em conta a variedade de elementos de cada *layout*, constituiu igualmente um desafio, principalmente na identificação de qual seria o padrão de *design* que pudesse ser aplicável.

6.3 Trabalho Futuro

Como trabalho futuro, pretende-se incorporar uma métrica que meça o grau de satisfação do utilizador. É uma métrica importante, pois permitirá aferir o grau de satisfação do utilizador face à interface apresentada e gerada dinamicamente. Uma possibilidade, para capturar o grau de satisfação, poderá passar pela criação de um componente que permita interagir com os utilizadores, onde estes podem atribuir o seu grau de satisfação. No artigo [16], sugerem a recomendação do uso de técnicas de análise de satisfação de modo a perceber se as decisões tomadas relativas à adaptação da interface estão corretas. Uma forma de recolher o grau de satisfação, seria aplicar *feedback* baseado em *emojis*, que visa expressar os graus de satisfação entre os utilizadores finais através da escolha de um *emoji* classificando a sua experiência de utilizador.

Uma questão fraturante neste tema de exploração de aplicações adaptáveis, prende-se com a abordagem mais adequada a adotar no primeiro contacto, e expetativas futuras que podem ser criadas. Uma alteração de interface de forma abrupta, apesar de poder ser o mais ajustado do ponto de vista lógico, poderá criar desconfiança e barreiras na interação com o sistema. Pode ser complexo para certos utilizadores, que estão "habituados" à noção que a interface permanece com o mesmo "aspeto" todos os dias, e num dia, aparecer "diferente". Deste modo, a abordagem deve ser moderada e iterativa. Sugerindo-se a adoção de uma funcionalidade de personalização, onde os utilizadores podem selecionar se estão recetivos a interfaces personalizadas, bem como o grau de alteração que estariam disponíveis a aceitar.

Atualmente o sistema utiliza uma API de alto nível de estimadores, disponibilizada pela TensorFlow, que fornece modelos de ML/DL pré-construídos. No entanto, quando o sistema ficar mais complexo, ou o conjunto de dados aumentar, o ideal seria criar o modelo utilizando estimadores personalizados de modo a ser possível configurar todos os parâmetros do

modelo a fim de se otimizar ainda mais o modelo e também incorporar a computação paralela, para se obter melhor desempenho.

Por fim e de forma lateral, é recomendado como trabalho futuro, que o princípio, métricas e algoritmos utilizados para promoverem a seleção de interfaces mais adaptáveis, seja analisada do ponto de vista da "Teoria do Evolucionismo".

6.4 Considerações Finais

Os sistemas atuais têm na sua maior preocupação e fator central o utilizador e a forma como se relacionam com estes. Desta forma, objetivos, como aumentar a experiência de satisfação, capacidade de retenção, ou fazer chegar a informação da forma mais simples para um entendimento mais rápido, são características distintivas dos sistemas mais evoluídos que se encontram a ser desenvolvidos.

Claramente que a solução apresentada possui margem de desenvolvimento e melhorias, sobretudo, por explorar uma área que se encontra a dar os primeiros passos, mas considerando a importância de ajustar o canal ao destinatário, será o grande próximo ciclo evolutivo, principalmente no que diz respeito a interfaces *web*.

Após toda a análise e conclusões, é possível perceber que abordagem de sistemas adaptáveis aos utilizadores é o caminho a seguir. Tendo em conta todos os recursos disponibilizados pelas *frameworks* de ML no mercado e as vantagens que ML pode oferecer nesta área de deteção de padrões é inevitável incluir esta inteligência de forma a criar o canal de excelência das aplicações de amanhã.

Por último, referir que durante o desenvolvimento desta dissertação, e perante os primeiros resultados que foram apresentados, os orientadores da Altice Labs, sendo da opinião que o estudo em questão, bem como a abordagem seria inovador na área, incentivaram que fosse tentado o registo de patente. O processo, encontra-se na fase de registo da patente provisória.

Bibliografia

[1] C. Papatheodorou, “Machine Learning in User Modeling.”

[2] F. Gullà, S. Ceccacci, M. Germani, and L. Cavalieri, “Design Adaptable and Adaptive User Interfaces: A Method to Manage the Information,” 2015, pp. 47–58.

[3] J.-J. Lee, R. Mccartney, and E. Santos, “Learning and Predicting User Behavior for Particular Resource Use.”

[4] M. P. Papazoglou, “AGENT-ORIENTED TECHNOLOGY IN SUPPORT OF E-BUSINESS,” *Commun. ACM*, vol. 44, no. 4, 2001.

[5] “Applications Of Machine Learning For Designers,” *Smashing Magazine*. [Online]. Available: https://www.smashingmagazine.com/2017/04/applications-machine-learning-designers/. [Accessed: 25-Mar-2018].

[6] J. Ben Schafer, J. A. Konstan, and J. Riedl, “E-Commerce Recommendation Applications,” *Data Min. Knowl. Discov.*, vol. 5, no. 1/2, pp. 115–153, 2001.

[7] “Google Fit.” [Online]. Available: https://www.google.com/fit/. [Accessed: 29-May-2018].

[8] “Music for everyone - Spotify.” [Online]. Available: https://www.spotify.com/pt/. [Accessed: 30-Jun-2018].

[9] “Netflix Portugal – Veja séries de televisão e filmes online.” [Online]. Available: https://www.netflix.com/pt/. [Accessed: 30-Jun-2018].

[10] R. Mayrhofer, H. Radi, and A. Ferscha, “RECOGNIZING AND PREDICTING CONTEXT BY LEARNING FROM USER BEHAVIOR.”

[11] P. Langley, “Machine Learning for Adaptive User Interfaces.”

[12] “LiftIgniter raises $6.4M to bring website personalization to the rest of the internet | TechCrunch,” 2017. [Online]. Available: https://techcrunch.com/2017/08/17/liftigniter-raises-6-4m-to-bring-website-personalization-to-the-rest-of-the-internet/. [Accessed: 24-Feb-2018].

[13] “E-Commerce Machine Learning | Machine learning conversion | Machine Learning Personalization - Liftigniter | LiftIgniter.” [Online]. Available: https://www.liftigniter.com/how-it-works/. [Accessed: 24-Feb-2018].

[14] V. Alvarez-Cortes, V. H., J. A. Ramirez Uresti, and B. E., “Current Challenges and Applications for Adaptive User Interfaces,” in *Human-Computer Interaction*, InTech, 2009.

[15] D. W. Cooper, F. P. Veitch, M. M. Anderson, and M. J. Clifford, “Adaptive diagnostics and personalized technical support (ADAPTS),” in *1999 IEEE Aerospace Conference. Proceedings (Cat. No.99TH8403)*, 1999, pp. 139–149 vol.3.

[16] N. Mezhoudi, “User Interface Adaptation based on User Feedback and Machine Learning.”

[17] J. Eisenstein and A. Puerta, “Adaptation in Automated User-Interface Design.”

[18] P. Ongsulee, “Artificial Intelligence, Machine Learning and Deep Learning,” *Fifteenth Int. Conf. ICT Knowl. Eng.*, pp. 1–6, 2017.

[19] K. G. Kim, “Deep Learning Book Review,” *Heal. Inf. Res*, vol. 2222351, no. 44, pp. 351–354, 2016.

[20] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Third Edition. Pearson, 2010.

[21] S. Russel, “Inteligência Artificial Introdução,” vol. 1.

[22] S. Wang, F. Ren, and H. Lu, “A review of the application of natural language processing in clinical medicine,” *2018 13th IEEE Conf. Ind. Electron. Appl.*, pp. 2725–2730, 2018.

[23] R. Sousa, “Serviços IoT: a magia da previsão de comportamentos,” *SPARK2D*, 2017. [Online]. Available: http://spark2d.com/servicos-iot-a-magia-da-previsao-de-comportamentos/. [Accessed: 20-Feb-2018].

[24] A. Talwar and Y. Kumar, “Machine Learning: An artificial intelligence methodology,” *Int. J. Eng. Comput. Sci.*, vol. 2, no. 12, 2013.

[25] R. Bissi, “A evolução do Machine Learning - Inteligência Artificial,” 2017. [Online]. Available: https://canaltech.com.br/inteligencia-artificial/a-evolucao-do-machine-learning-102120/. [Accessed: 19-Feb-2018].

[26] T. M. Mitchell, *Machine Learning*. 1997.

[27] J. J. Valletta, C. Torney, M. Kings, A. Thornton, and J. Madden, “Applications of machine learning in animal behaviour studies,” *Anim. Behav.*, vol. 124, pp. 203–220, 2017.

[28] “Training and Testing Data Sets,” *Microsoft Docs*, 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/training-and-testing-data-sets. [Accessed: 20-Feb-2018].

[29] M. Smith, “Preparing For The Future Of Artificial Intelligence,” *Natl. Sci. Technol. Counc.*, 2016.

[30] D. Matos, “Conceitos Fundamentais de Machine Learning,” *Ciência e Dados*. [Online]. Available: http://www.cienciaedados.com/conceitos-fundamentais-de-machine-learning/. [Accessed: 19-Feb-2018].

[31] A. Wosiak, A. Zamecznik, and K. Niewiadomska-Jarosik, “Supervised And Unsupervised Machine Learning For Improved Identification Of Intrauterine Growth Restriction Types,” *2016 Fed. Conf. Comput. Sci. Inf. Syst.*, vol. 8, pp. 323–329, 2016.

[32] “Introduction to Deep Neural Networks (Deep Learning),” *Deeplearning4j: Open-source, Distributed Deep Learning for the JVM*. [Online]. Available: https://deeplearning4j.org/neuralnet-overview. [Accessed: 14-May-2018].

[33] “The Evolution and Core Concepts of Deep Learning & Neural Networks,” *Analytics Vidhya*. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/08/evolution-core-concepts-deep-learning-neural-networks/. [Accessed: 18-May-2018].

[34] L. Rampasek and A. Goldenberg, “TensorFlow: Biology’s Gateway to Deep Learning?,” *Cell Syst.*, vol. 2, no. 1, pp. 12–14, 2016.

[35] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” 2016.

[36] P. Goldsborough, “A Tour of TensorFlow Proseminar Data Mining.”

[37] H.-T. Cheng, Z. Haque, L. Hong, M. Ispir, C. Mewald, I. Polosukhin, G. Roumpos, D. Sculley, J. Smith, D. Soergel, Y. Tang, P. Tucker, M. Wicke, C. Xia, and J. Xie, “TensorFlow Estimators: Managing Simplicity vs. Flexibility in High-Level Machine Learning Frameworks,” pp. 1763–1771, 2017.

[38] A. Parvat, J. Chavan, S. Kadam, S. Dev, and V. Pathak, “A survey of deep-learning frameworks,” *Proc. Int. Conf. Inven. Syst. Control. ICISC 2017*, pp. 1–7, 2017.

[39] A. Vishnu, C. Siegel, and J. Daily, “Distributed TensorFlow with MPI.”

[40] “An open-source machine learning framework for everyone,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/. [Accessed: 21-Feb-2018].

[41] “TensorFlow Framework and GPU Acceleration,” *NVIDIA Data Center*. [Online]. Available: https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/tensorflow/. [Accessed: 15-May-2018].

[42] “Installing TensorFlow on Windows,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/install/install_windows. [Accessed: 15-May-2018].

[43] V. Rao, “Getting started with TensorFlow,” *IBM*. [Online]. Available: https://www.ibm.com/developerworks/library/cc-get-started-tensorflow/. [Accessed: 21-Feb-2018].

[44] H. Yi, H. Jung, and S. Bae, “Deep Neural Networks for Traffic Flow Prediction,” *2017 Ieee Int. Conf. Big Data Smart Comput.*, pp. 328–331, 2017.

[45] “Estimators,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/programmers_guide/estimators. [Accessed: 15-May-2018].

[46] S. Pölsterl’s, “Denoising Autoencoder as TensorFlow estimator.” [Online]. Available: https://k-d-w.org/blog/103/denoising-autoencoder-as-tensorflow-estimator. [Accessed: 15-May-2018].

[47] “Creating Custom Estimators,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/get_started/custom_estimators. [Accessed: 15-May-2018].

[48] “Premade Estimators,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/get_started/premade_estimators. [Accessed: 15-May-2018].

[49] J. Lewis, “Microservices,” 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html#CharacteristicsOfAMicroserviceArchitecture. [Accessed: 03-Oct-2017].

[50] M. Fowler, “MicroservicePremium,” 2015. [Online]. Available: https://martinfowler.com/bliki/MicroservicePremium.html. [Accessed: 03-Oct-2017].

[51] “TECHNOLOGIES - Think Micro,” *Skava*. [Online]. Available: https://www.skava.com/technologies. [Accessed: 27-May-2018].

[52] “Skava.” [Online]. Available: http://www.skava.com/. [Accessed: 27-May-2018].

[53] H. Obaidy, “What are microservices and why you should care,” *Skava Blog*, 2018. [Online]. Available: https://blog.skava.com/2018/02/15/what-are-microservices-and-why-you-should-care/. [Accessed: 27-May-2018].

[54] M. Fowler, “Microservice Trade-Offs,” *MartinFowler.com*, 2015. [Online]. Available: https://martinfowler.com/articles/microservice-trade-offs.html#summary. [Accessed: 27-May-2018].

[55] David Ramel, “3 Leading Enterprise Low-Code App Development Platforms Compared,” 2017. [Online]. Available: https://adtmag.com/blogs/dev-watch/2017/05/low-code-tools.aspx. [Accessed: 19-Dec-2017].

[56] “About OutSystems - Facts and Figures.” [Online]. Available: https://www.outsystems.com/company/about/. [Accessed: 19-Dec-2017].

[57] “3 Leading Enterprise Low-Code App Development Platforms Compared - - ADTmag,” *2017-05-19*. [Online]. Available: https://adtmag.com/blogs/dev-watch/2017/05/low-code-tools.aspx. [Accessed: 24-Oct-2017].

[58] “What is Salesforce? Cloud CRM Solutions.” [Online]. Available: https://www.salesforce.com/eu/products/what-is-salesforce/. [Accessed: 19-Dec-2017].

[59] “Mendix: We help enterprises achieve their digital goals.” [Online]. Available: https://www.mendix.com/company/. [Accessed: 19-Dec-2017].

[60] H. Obaidy, “Skava Announces SkavaSTUDIO™, Allowing Retailers to Quickly Create Outstanding Online Shopping and Marketing Experiences Without Requiring IT Resources - Skava BlogSkava Blog,” 2013. [Online]. Available: https://blog.skava.com/2013/10/01/skava-announces-skavastudio-allowing-retailers-to-quickly-create-outstanding-online-shopping-and-marketing-experiences-without-requiring-it-resources/. [Accessed: 27-May-2018].

[61] “About Us | Wix.com.” [Online]. Available: https://www.wix.com/about/us. [Accessed: 26-Jun-2018].

[62] “Inside the Wix Code App Development Platform.” [Online]. Available: https://www.upwork.com/hiring/development/wix-code/. [Accessed: 27-Jun-2018].

[63] D. Riehle and T. Gross, “Role Model Based Framework Design and Integration,” pp. 117–133, 1998.

[64] J. O’Neill, “Web Components VS Frameworks,” *Medium*. [Online]. Available: https://medium.com/@oneeezy/frameworks-vs-web-components-9a7bd89da9d4. [Accessed: 22-Nov-2017].

[65] P. De Ryck, N. Nikiforakis, L. Desmet, F. Piessens, and W. Joosen, “Protected Web Components: Hiding Sensitive Information in the Shadows,” *IEEE Comput. Soc.*, 2015.

[66] D. Cooney, “Introduction to Web Components,” 2013. [Online]. Available: https://www.w3.org/TR/2013/WD-components-intro-20130606/. [Accessed: 25-Sep-2017].

[67] V. Reis, “Entenda de uma vez por todas o que é React.JS, Angular 2, Aurelia e Vue.JS,” 2016. [Online]. Available: https://medium.com/by-vinicius-reis/o-que-e-react-ng2-auleria-vue-e34b0c77b5a1. [Accessed: 25-Sep-2017].

[68] Y. Gu, C. Xu, and M. Zheng, “Using React Native in an Android App.”

[69] A. Fedosejev, *React.js Essentials A fast-paced guide to designing and building scalable and maintainable web apps with React.js*. Packt Publishing, 2015.

[70] M. Duta, “The battle of Frontend Frameworks (Angular, React, Vue.js),” 2017. [Online]. Available: https://mduta.com/blog/battle-frontend-frameworks-angular-react-vuejs. [Accessed: 19-Sep-2017].

[71] E. Holmes and T. Bray, *Getting Started with React Native*. Packt Publishing, 2015.

[72] “Angular - AngularJS to Angular Concepts: Quick Reference.” [Online]. Available: https://angular.io/guide/ajs-quick-reference. [Accessed: 24-Jun-2018].

[73] “Getting Started - vue.js.” [Online]. Available: https://012.vuejs.org/guide/. [Accessed: 23-Nov-2017].

[74] “Introduction — Vue.js.” [Online]. Available: https://vuejs.org/v2/guide/. [Accessed: 22-Sep-2017].

[75] A. Syromiatnikov and D. Weyns, “A Journey through the Land of Model-View-Design Patterns,” in *2014 IEEE/IFIP Conference on Software Architecture*, 2014, pp. 21–30.

[76] J. Neuhaus, “Angular vs. React vs. Vue: A 2017 comparison – unicorn.supplies – Medium,” 2017. [Online]. Available: https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176. [Accessed: 27-Sep-2017].

[77] “TypeScript - JavaScript that scales.” [Online]. Available: https://www.typescriptlang.org/. [Accessed: 23-Nov-2017].

[78] A. Stringfellow, “TypeScript vs. JavaScript: Should You Migrate Your JavaScript Coding Project to TypeScript? - DZone Web Dev,” 2017. [Online]. Available: https://dzone.com/articles/typescript-vs-javascript-should-you-migrate-your-j. [Accessed: 28-Sep-2017].

[79] Y. Fain and A. Moiseev, *Angular 2 development with TypeScript*. 2016.

[80] H. Brendon, “AngularJS series: Entendendo o framework – Medium,” 2016. [Online]. Available: https://medium.com/@hudsonbrendon/angularjs-series-entendendo-o-framework-1499a841d51f. [Accessed: 12-Sep-2017].

[81] Max, “Angular vs React.js vs Vue.js,” 2017. [Online]. Available: https://academind.com/articles/javascript/angular-vs-reactjs-vs-vuejs/. [Accessed: 19-Sep-2017].

[82] “Angular - Angular Dependency Injection.” [Online]. Available: https://angular.io/guide/dependency-injection. [Accessed: 25-Jun-2018].

[83] “Angular - Lazy Loading Feature Modules.” [Online]. Available: https://angular.io/guide/lazy-loading-ngmodules#lazy-loading-feature-modules. [Accessed: 26-Jun-2018].

[84] “Angular - The Ahead-of-Time (AOT) Compiler.” [Online]. Available: https://angular.io/guide/aot-compiler. [Accessed: 25-Jun-2018].

[85] “Angular CLI.” [Online]. Available: https://cli.angular.io/. [Accessed: 23-Nov-2017].

[86] “Criando aplicações Angular com Angular CLI,” 2017. [Online]. Available: http://blog.alura.com.br/criando-aplicacoes-angular-com-angular-cli/. [Accessed: 23-Nov-2017].

[87] “Angular - Architecture Overview.” [Online]. Available: https://angular.io/guide/architecture. [Accessed: 03-Sep-2017].

[88] “Angular - Introduction to components,” *Angular*. [Online]. Available: https://angular.io/guide/architecture-components#data-binding. [Accessed: 18-May-2018].

[89] S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to Build High-Performance Network Programs,” *IEEE Internet Comput.*, vol. 14, no. 6, pp. 80–83, Nov. 2010.

[90] “About | Node.js.” [Online]. Available: https://nodejs.org/en/about/. [Accessed: 27-Sep-2017].

[91] “01 - What is npm? | npm Documentation,” 2017. [Online]. Available: https://docs.npmjs.com/getting-started/what-is-npm. [Accessed: 22-Sep-2017].

[92] “Getting Started | Yarn.” [Online]. Available: https://yarnpkg.com/en/docs/getting-started. [Accessed: 27-Sep-2017].

[93] J. Ooms, “Embedded Scientific Computing: A Scalable, Interoperable and Reproducible Approach to Statistical Software for Data-Driven Business and Open Science.” 01-Jan-2014.

[94] “Working with package.json,” *npm Documentation*. [Online]. Available: https://docs.npmjs.com/getting-started/using-a-package.json. [Accessed: 26-Jun-2018].

[95] “Yarn.” [Online]. Available: https://www.npmjs.com/package/yarn. [Accessed: 26-Jun-2018].

[96] “Web API for MongoDB.” [Online]. Available: https://softinstigate.atlassian.net/wiki/spaces/RH/overview. [Accessed: 29-Sep-2017].

[97] “Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST).” [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Accessed: 23-Jul-2018].

[98] M. Fowler, “Richardson Maturity Model,” 2010. [Online]. Available: https://martinfowler.com/articles/richardsonMaturityModel.html. [Accessed: 04-Oct-2017].

[99] P. Adamczyk, P. H. Smith, R. E. Johnson, and M. Hafiz, “REST and Web Services: In Theory and in Practice,” in *REST: From Research to Practice*, New York, NY: Springer New York, 2011, pp. 35–57.

[100] X. Ming and X. Xinping, “A Comparative Analysis of Discretization Algorithms for Data Mining,” pp. 1434–1438, 2009.

[101] R. Prati, “Data Mining - A Preparação dos Dados.”

[102] “tf.metrics.accuracy | TensorFlow.” [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/metrics/accuracy. [Accessed: 25-Jul-2018].